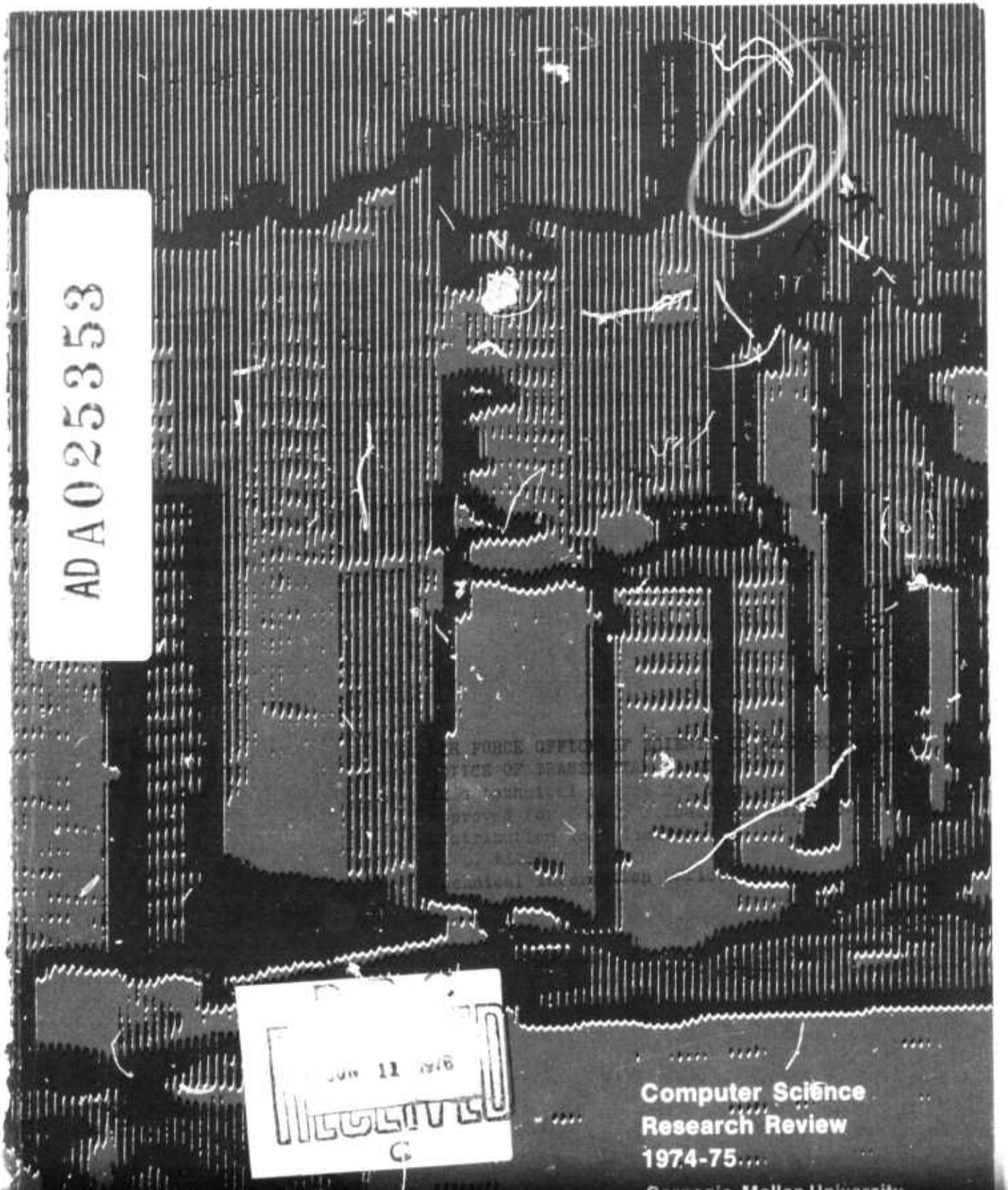


Computer Science Research Review 1974-75

Carnegie-Mellon University

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR - TR - 76 - 0605	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: Annual rept.	
4. TITLE (and Subtitle): COMPUTER SCIENCE RESEARCH REVIEW 1974-75	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Department of Computer Science	8. CONTRACT OR GRANT NUMBER(s): F44620-73-C-0074, N00014-67-A-0314-0010
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University, Computer Science Dept., Pittsburgh, PA 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61101D, AO 2466
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research (NM), Bolling AFB, DC 20032	12. REPORT DATE: Aug 75; 13. NUMBER OF PAGES: 75 (78p.)
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Defense Advanced Research Project Agency, 1400 Wilson Blvd, Arlington, VA 22209	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

16 ARPA Order - 2466

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A number of themes which run through the papers in this research review are present elsewhere in the research. One theme is decomposition, a second theme is experimentation and a third theme is realism. This review contains four papers and an annual report of the Computer Science Department, Carnegie-Mellon University.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

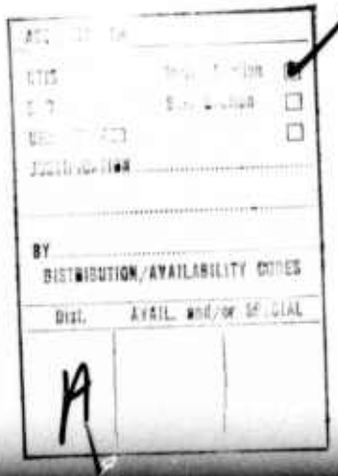
403 085

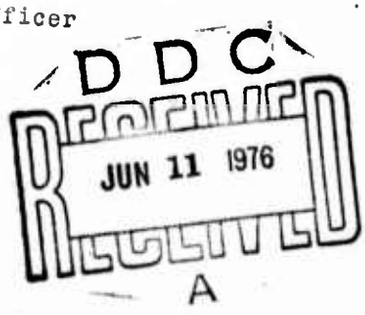
AFOSR - TR - 76 - 0605

(6)

Approved for public release;
distribution unlimited.

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is approved for public release IAW AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer





Computer Science Research Review
1974-75

An Annual Report
published by the
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

Graphics by Darlene Beachley
Printed by Hoechstetter Printing Co.

The work reported here was largely supported by the
Advanced Research Projects Agency
of the Office of the Secretary of Defense
(Contract number F44620-73-C-0074)
and is monitored by the
Air Force Office of Scientific Research.
This work was also supported in part by
the National Science Foundation (Contract numbers
GJ 32111, GJ 32758, DCR 74-04187, DCR 75-07251,
DCR 74-24573, GJ 32259),
the Office of Naval Research (Contract numbers
N00014-67-A-0314-0010, N00014-67-A-0314-0018),
and numerous private grants for which
we are deeply grateful. For a complete listing
of these grants and contracts, see page 70
of this report.

Pittsburgh, Pennsylvania
September, 1975

Contents

Annual Review Introduction

A number of themes which run through the papers in this Research Review are present elsewhere in our research.

One theme is decomposition. The decomposition of algorithms and problems leads to new algorithms for parallel machines. On the other hand, for certain problems there are theoretical limits on decomposition, leading to theoretical limits on speed-up for synchronous or asynchronous parallel machines. In speech understanding research, decomposition permits processing by various knowledge sources and facilitates implementation on our asynchronous multi-processor machine. We see this as a cost-effective way of obtaining large amounts of computing power.

A second theme is experimentation. In computer science we often build a system so we can manipulate and study it. This approach helps us to incorporate the latest technology in our design efforts. The system must enjoy high performance to permit enough realistic experiments.

A third theme is realism. We are concerned with constructing as realistic models as possible, limited only by our current understanding of the problem and the power of our analytical tools. For example, this is basic in our approach to the analysis of algorithms and complexity.

I want to turn next to some very pleasant news. Individual members of the Department have received outstanding recognition. Al Newell and Herb Simon share the 1975 Turing Award for their joint scientific efforts extending over twenty years during which they made basic contributions to artificial intelligence, the psychology of human cognition, and list processing. Gordon Bell has won the 1975 McDowell Award for outstanding contributions in the areas of technical design, education, and publications influential in developing the computer field. Raj Reddy has won a Guggenheim Fellowship to work on functionally specialized computer architectures for speech and vision problems. Jack Buchanan serves for a year as a Judicial Fellow with the U.S. Supreme Court and the Federal Judiciary Center to aid in modernizing the administration of the Federal courts.

In October 1975, the Department celebrates its tenth anniversary. Present and past members of the Department will join in the celebration, a three day technical symposium on research at the frontiers of computer science. Invited and selected papers will appear in a commemorative volume.

J. F. T.

Bounds on the Speed-up of Parallel Evaluation of Recurrences

Laurent Hyafil and H. T. Kung

I. Introduction

To understand the performance of parallel computers such as ILLIAC IV and C.mmp, we must know the largest speed-up that can be obtained for a *given* task. If there are k processors, the largest speed-up that can be achieved is k and we call this o*ptimal* speed-up. The speed-up in general depends on the parallel decomposition of a particular computing task and the various aspects of the multiprocessing system, including memory contention, process communication, operating system overhead, etc. In this paper, we concentrate on the issue of decomposing tasks, and assume that the multiprocessing system is idealized so that it causes no delays at all. We shall show that even under this idealized assumption, there are problems for which, because the parallel decompositions are inherently difficult, the optimal speed-up can not be achieved.

This paper studies bounds on speed-ups for a particular problem, i.e., the problem of evaluating (or solving) recurrences, which is defined as follows:

Input: $x_0, x_{-1}, \ldots, x_{-p+1}$ and rational functions r_i, $i \geq 1$.

Output: x_n, which is defined by

$$x_i = r_i(x_{i-1}, \ldots, x_{-p+1}), \; i \geq 1.$$

Since the x_i are defined iteratively, the problem appears on the surface to be highly serial. Hence it is interesting to investigate how parallel algorithms can be designed and what are the theoretical limits of using parallelism for the problem. We consider the recurrence problem also because it is important in practice and is simply stated so that we might obtain some insight into the nature of parallel computation by studying it. We shall survey a number of results in connection with bounds on the speed-ups of parallel evaluation of various kinds of recurrences, especially when the size n of the problem is large, or when $n \to \infty$. For simplicity we assume that each arithmetic operation takes one unit of time. Consider a k-processor machine. We shall see, for example, that the speed-up for the first order linear recurrence problem is at most (2/3)k + (1/3) even under the idealized assumption. Of course, the actual speed-up obtained from a real k-processor machine would be $\leq$ (2/3)k + (1/3). The difference between (2/3)k + (1/3) and k is rather significant. For example, if K = 16, 64, then the speed-up for the problem is at most 11, 43, respectively, no matter how efficient the k-processor machine is. The reason that we get at most 70 percent of the speed-up we might expect for the problem is the inherent dependence of variables in the recurrence. Nonlinear recurrences are even worse. It is shown that the speed-ups for a certain class of nonlinear recurrence problems are always bounded by a constant no matter how many processors are used and how large the size of the problem is. Hence the dependency relationships within the variables of these nonlinear recurrences are even stronger. We believe that the study of these dependency relationships is fundamental for understanding parallel computation.

The kind of results which are to be presented in the paper could be useful in the following two ways. First, the theoretical bounds on speed-ups provide grounds for testing the efficiencies of algorithms and the multiprocessing system. (For example, it would be very helpful if tight theoretical bounds on speed-ups are known for benchmark tasks.) Second, the constructions of the algorithms designed for the idealized machine are instructive and often lead to useful insights into the nature of designing efficient algorithms for real machines.

2. Definitions and Notation

An algorithm for evaluating x_n is defined to be a directed acyclic graph in a natural way. For example, the graph of Figure 1 defines a parallel algorithm using three processors for evaluating x_3, which is defined by

$$x_0 = a_1,$$
$$x_i = b_i x_{i-1} + a_{i+1}, \; i = 1,2,3.$$

(Note that $x_3 = ((a_1 b_1 + a_2) b_2 + a_3) b_3 + a_4$.)

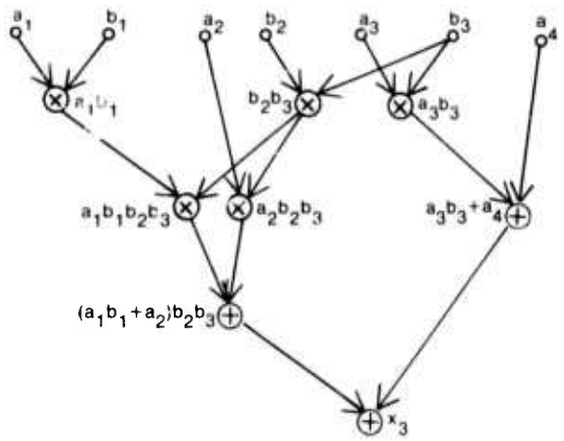


Figure 1

Consider the directed graph which defines an algorithm. We define the depth of the graph to be the *time*, define

$T_k(x_n)$ = minimum time needed to evaluate x_n by an algorithm using k processors.

and define the *speed-up* of the problem of evaluating x_n by using k processors to be

$$S_k(x_n) = \frac{T_1(x_n)}{T_k(x_n)}.$$

(In Hyafil and Kung [75a] these definitions are given in a more rigorous way.)

By a simple simulation argument, one can easily see that $T_1(x_n) \le k\, T_k(x_n)$. Hence

$$S_k(x_n) \le k, \quad \forall k, \forall n.$$

k is a trivial upper bound on $S_k(x_n)$. Bounds smaller than k are nontrivial. We shall show some nontrivial upper bounds on $S_k(x_n)$ in the following sections.

3. First Order Linear Recurrences

A first order linear recurrence is defined by

$$(1)\quad x_i = a_i x_{i-1} + b_i,\ i \ge 1.$$

It is the most fundamental recurrence, in the sense that algorithms for solving it often form basic algorithms for solving other types of recurrences. The trivial algorithm which computes $x_1, x_2, \ldots, x_n$ iteratively according to (1) is the optimal sequential algorithm, since it takes time 2n and any algorithm has to take time at least 2n for using all the inputs. Hence

$$(2)\quad T_1(x_n) = 2n.$$

The algorithm, however, is not suitable for parallel computers because it does not provide any parallelism. New algorithms are needed for parallel computers. Various parallel algorithms have been developed by many people, including Brent [70, 74], Kogge [74], Kogge and Stone [73], Kuck and Maruyama [73], Kuck and Maruyama [75], Lambiotte and Voigt [74],Stone [73, 74] and Winograd [74]. The basic idea of these algorithms can be explained as follows:

Note that (1) is equivalent to

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ 1 \end{bmatrix},\ i \ge 1.$$

Hence

$$(3)\quad \begin{bmatrix} x_n \\ 1 \end{bmatrix} = \left(\prod_{1}^{n} \begin{bmatrix} a_i & b_i \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x_0 \\ 1 \end{bmatrix},$$

which can clearly be computed in parallel. Using the fact that the multiplication of two matrices of the form $\begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix}$ takes three operations and results in a matrix of the same form, while the multiplication of $\begin{bmatrix} x & x \\ 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} x \\ 1 \end{bmatrix}$ uses two operations and results in a vector of the form $\begin{bmatrix} x \\ 1 \end{bmatrix}$, in Hyafil and Kung [75b] a parallel algorithm based on (3) is derived and establishes that

$$(4)\quad T_k(x_n) \le \frac{3n}{k+1/2} + c_1 \log k$$

for some constant $c_1 > 0$. (4) is an improvement over the corresponding result in Winograd [74] when n is large and k is fixed.

In Hyafil and Kung [75a] it is shown that if an algorithm computes x_n in time t with w operations, then

(5) $w \geq 3n - \frac{t}{2}$.

Suppose that $t < T_1(x_n) = 2n$. Then by (5) $w > 2n$. Hence if a parallel algorithm is faster than the optimal sequential algorithm, then it must perform more operations than the sequential algorithm. This turns out to be the basic reason why the optimal speed-up cannot be achieved for the problem. Indeed, lower bounds on $T_k(x_n)$ can be easily derived from (5) as follows. Suppose that k processors are used. Observe that for any algorithm, $kt \geq w$. Hence by (5) we have

(6) $T_k(x_n) \geq \frac{3n}{k+1/2}$, $\forall k, \forall n$.

Suppose that $w \geq 2^{\lfloor \log k \rfloor}$. (This is true when, say, $n \geq k$.) In Heller [75] it is pointed out that in this case by the same argument as used in Munro and Paterson [73, Theorem 1], the bound in (6) can be slightly improved. In fact, we have

$$K(t-\lfloor \log k \rfloor) + 2^{\lfloor \log k \rfloor} - 1 \geq w,$$

which together with (5) yields

$$T_k(x_n) \geq \frac{3n}{k+1/2} + (k\lfloor \log k \rfloor + 1 - 2^{\lfloor \log k \rfloor})/(k+1/2).$$

Hence

(7) $T_k(x_n) \geq \frac{3n}{k+1/2} + c_2 \log k - c_3$, $n \geq k$

for some constants $c_2 > 0$ and $c_3 > 0$. From (4) and (7), we know that the bounds are essentially sharp for $n \geq k$.

From (2), (4), (6) and (7) we have the following

Theorem 1

For the first order linear recurrence defined by (1),

(8) $$\leq S_k(x_n) \leq \begin{cases} (2/3)k + 1/3, & \forall k, \forall n \\ \dfrac{2k+1}{3 + \dfrac{c_1(k+1/2)\log k}{n}}, & \forall k, \forall n \geq k. \end{cases}$$

(9)

The upper bound in (8) implies that even for the simplest recurrence defined by (1), we can get at most 70 percent of the optimal speed-up.

The algorithm used to establish the bound in (4) can be extended to solve first order *vector* linear recurrences, defined by

(10) $\underline{x}_i = A_i \underline{x}_{i-1} + \underline{b}_i$, $i \geq 1$,

where the $\underline{x}$'s and the $\underline{b}$'s are p-vectors and the A's pxp matrices. The upper bounds on time for solving these vector recurrences can similarly be obtained.

4. Pth Order Linear Recurrences

A pth order linear recurrence is defined by

(11) $x_i = \sum_{j=i-p}^{i-1} a_{ij} x_j + b_i$, $i \geq 1$.

The problem for solving such a recurrence in parallel has been considered in Chen and Kuck [75], Kogge [74] and Kogge and Stone [73].

The following theorem generalizes the upper bound result in (8).

Theorem 2 (Hyafil and Kung [75b])

For the pth order linear recurrence defined by (11),

(12) $s_k(x_n) \leq \frac{2p}{2p+1} k + c_4$, $\forall p, \forall k, \forall n$.

for some constant c_4.

Since $\frac{2p}{2p+1} < 1$, the theorem implies that we cannot essentially obtain the optimal speed-up for solving pth order linear recurrences for any p, when k is large.

We now consider parallel algorithms for solving the recurrence defined by (11). The idea is to convert it into a first order vector linear recurrence of the form (10), which can then be solved by algorithms used in the preceding section.

The naive approach for the conversion would be the following way: Define vectors

$$\underline{x}_i = \begin{bmatrix} x_i \\ x_{i-1} \\ \vdots \\ x_{i-p+1} \end{bmatrix}, \quad i \geq 0$$

then (11) is equivalent to

$$(13)\quad \underline{x}_i = A_i \underline{x}_{i-1} + \underline{b}_i,\ i = 1,2,\ldots,n$$

where the A_i are certain companion matrices. Then algorithms for solving first order vector linear recurrences can be applied to compute $\underline{x}_n$ (and hence x_n) from (13). We shall use another conversion technique, which will lead us to p times faster algorithms for the case that k, p are fixed and $n \to \infty$. The idea is explained in the following for the case of p = 3. We can write a 3rd order linear recurrence as

$$x_i = -\sum_{j=i-3}^{i-1} \bar{a}_{ij} x_j + b_i,\ i \geq 1$$

where $\bar{a}_{ij} = -a_{ij}$. Then for computing, say x_6, from x_0, x_{-1}, x_{-2} we have

$$\left[\begin{array}{ccc|ccc|ccc} \bar{a}_{1,-2} & \bar{a}_{1,-1} & \bar{a}_{10} & 1 & & & & & \\ \bar{a}_{2,-1} & \bar{a}_{20} & & \bar{a}_{21} & 1 & & & & \\ & \bar{a}_{30} & & \bar{a}_{31} & \bar{a}_{32} & 1 & & & \\ \hline & & & \bar{a}_{41} & \bar{a}_{42} & \bar{a}_{43} & 1 & & \\ & & & & \bar{a}_{52} & \bar{a}_{53} & \bar{a}_{54} & 1 & \\ & & & & & \bar{a}_{63} & \bar{a}_{64} & \bar{a}_{65} & 1 \end{array}\right] \left[\begin{array}{c} x_{-2} \\ x_{-1} \\ x_0 \\ \hline x_1 \\ x_2 \\ x_3 \\ \hline x_4 \\ x_5 \\ x_6 \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ \hline b_4 \\ b_5 \\ b_6 \end{array}\right]$$

If we partition the matrix and vectors into blocks as indicated above, then we have

$$\begin{bmatrix} A_1 & T_1 & \bigcirc \\ \bigcirc & A_2 & T_2 \end{bmatrix} \begin{bmatrix} \underline{x}_0 \\ \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \end{bmatrix}.$$

Hence

$$\underline{x}_1 = -T_1^{-1} A_1 \underline{x}_0 + T_1^{-1} \underline{b}_1,$$

$$\underline{x}_2 = -T_2^{-1} A_2 \underline{x}_1 + T_2^{-1} \underline{b}_2,$$

which is a first order vector linear recurrence. Using the same idea, for general p, we have

$$(14)\quad \underline{x}_i = (T_i^{-1} A_i)\underline{x}_{i-1} + T_i^{-1} \underline{b}_i,\ I = 1,2,\ldots,m$$

where $m = \lceil n/p \rceil$, T_i, A_i, $\underline{x}_i$, $\underline{b}_i$ are of size p, and T_i are triangular. We shall first compute $T_i^{-1} A_i$ and $T_i^{-1} \underline{b}_i$ for $i \leq m$, and then use algorithms in Section 3 to solve the recurrence (14). Since $m = \lceil n/p \rceil$, the recurrence (14) is shorter than the recurrence (13) by a factor of p. Thus we get faster algorithms. (It turns out that the cost of computing $T_i^{-1} A_i$ and $T_i^{-1} \underline{b}_i$ is not crucial.) From this approach it is immediate to prove that

$$(15)\quad T_k(x_n) \leq c_5 \left(\frac{p^\alpha}{k} n + p^\alpha \log n\right),$$

for some constant $c_5 > 0$, where $\alpha = 2$ when the usual matrix multiplication algorithm is used and $\alpha = 1.82$ when the Strassen's matrix multiplication algorithm (Strassen [69]) is used. (In Hyafil and Kung [75b] it is shown that the bound in (15) also holds for the problem of solving nxn *band* linear system with bandwidth p.) Since $T_1(x_n) \geq (p+1)n$, taking $\alpha = 1.82$ in (15) we have that for any k and p,

$$(16)\quad S_k(x_n) \geq \frac{1}{c_5} \left(\frac{k}{p^{.82}}\right),\ \text{as } n \to \infty,$$

for the problem of solving pth order linear recurrences. Does S_k/k indeed decrease as p increases? The question is still open. We only know that by (12) S_k is always less than k for large k. We believe that as p increases, more dependency relationships on the x's defined by (11) will be introduced and hence S_k/k will decrease.

Conjecture

Consider the problem of solving pth order linear recurrences defined by (11). Let the maximal speed-up ratio achievable by using k processors to be

$$\bar{S}_k(p) = \max_n S_k(x_n).$$

Then there exists a monotonically decreasing function λ such that

$$\bar{S}_k(p) \leq \lambda(p)k,\ \forall k,$$

and $\lambda(p) \to 0$ as $p \to \infty$.

The following theorem relates our conjecture on speed-ups to the matrix multiplication problem.

Theorem 3 (Hyafil and Kung [75b])

If the conjecture is true then $0(n^2/\lambda(n))$ is a lower bound on the number of arithmetic operations needed to multiply two nxn matrices.

Note that the question of whether or not matrix multiplication can be done in $0(n^2)$ operations has been open for some years.

5. General Linear Recurrences

A general linear recurrence is defined by

$$(17)\quad x_i = \sum_{j=0}^{i-1} a_{ij}x_j + b_i,\ i \geq 1.$$

The problem of solving general linear recurrences is reducible to that of solving triangular linear systems. Heller [74a] first considered the problem of solving (17) in parallel and gave algorithms which take time $O(\log^2 n)$ and use $O(n^4)$ processors. It was shown later that the problem in fact could be done in time $O(\log^2 n)$ with $O(n^3)$ processors by a number of people in at least three different ways (see, e.g., Borodin and Munro [75], Chen and Kuck [75], Heller [74b] and Orcutt [74]).

For the case of using small parallelism it is shown in Hyafil and Kung [74] that

$$(18)\quad T_k(x_n) \leq \frac{n^2}{k} + c_6 n \qquad \text{if } k \leq n,$$

$$T_k(x_n) \leq \begin{cases} c_7 n^{2-r}\log n & \text{if } k=\lfloor n^r \rfloor \text{ and } 1<r<3/2, \\ c_8 n^{1-r/3}\log^2 n & \text{if } k=\lfloor n^r \rfloor \text{ and } 3/2 \leq r<3. \end{cases}$$

where c_6, c_7, c_8 are positive constants.

Since there are $n(n+1)/2$ inputs for the recurrence (17), we have

$$T_1(x_n) \geq \frac{n(n+1)}{2} - 1,$$

while the trivial sequential algorithm establishes that

$$T_1(x_n) \leq n(n+1).$$

There is a gap between the lower and upper bounds on $T_1(x_n)$. We believe that $T_1(x_n) = n^2 + 0(n)$. Suppose that is true. Then from (18), we have $S_k(x_n) \to k$ as $n \to \infty$, i.e., optimal speed-up is achieved asymptotically, which would be in interesting contrast with pth order linear recurrences, where optimal speed-ups are not asymptotically achieved.

6. Nonlinear Recurrences

A nonlinear recurrence is defined by

$$(19)\quad x_i = \varphi(x_{i-1}, x_{i-2}, \ldots, x_{i-p}),\ i \geq 1,$$

where φ is a nonlinear rational function. Write $\varphi = \varphi_1/\varphi_2$ where φ_1 and φ_2 are polynomials which are relatively prime. Define the degree of a nonlinear recurrence to be

$$\deg \varphi = \max(\deg \varphi_1, \deg \varphi_2).$$

Hence, for example, the well known recurrence,

$$(20)\quad x_{i+1} = (1/2)(x_i + A/x_i),$$

for approximating $\sqrt{A}$ has degree 2. For linear recurrences we can have unbounded speed-up when $k \to \infty$ and $n \to \infty$. For example, by Theorem 1 we know that if $k = n$ the first order linear recurrence can be sped-up by a factor of $n/\log n$, which is unbounded as $n \to \infty$. The following theorem shows that the theory of nonlinear recurrences of degree > 1 is completely different from that of linear recurrences.

Theorem 4 (Kung [74])

For the recurrence defined by (19), if deg $\varphi > 1$, then

$$S_k(x_n) \leq c_9,\ \forall k, \forall n,$$

for some constant c_9.

The theorem implies that, e.g., the recurrence defined by (20) cannot essentially be sped up by using parallelism.

The only nonlinear recurrences which can possibly have unbounded speed-up by using parallelism are of the form

$$(21)\quad x_i = \left(\sum_{j=i-p}^{i-1} a_{ij}x_j + b_i\right) \Big/ \left(\sum_{j=i-q}^{i-1} c_{ij}x_j + d_i\right),$$

which is of degree one. Indeed, the recurrence

$$(22)\quad x_i = a_i + \frac{b_i}{x_{i-1}},$$

i.e., a continued fraction, can be sped up.

Theorem 5 (Hyafil and Kung [75b], Kogge [74], Kuck and Maruyama [73] and Winograd [74])

For the recurrence defined by (22),

$$(23)\quad (1/2)k + c_{10} \geq S_k(x_n) \geq \begin{cases} c_{11}\left(\frac{n}{\log n}\right) & \text{if } k \geq n, \\ (2/5)k \text{ as } n \to \infty, & \forall k. \end{cases}$$

for some constants c_{10}, c_{11}.

By Theorem 1 and (23) we note that recurrences with division seem to be more difficult than those without division in parallel computation. The same observation can also be made to the problem of evaluating arithmetic expressions (see Brent [74] and Winograd [74]).

It is clear that the recurrence

$$x_i = \frac{a_i x_{i-1} + b_i}{c_i x_{i-1} + d_i}$$

can also be sped up by using parallelism, since it can be transformed into a continued fraction. However, by the following theorem we know general recurrences defined by (21) cannot essentially be sped up.

Theorem 6 (Hyafil and Kung [75b])

For the recurrence defined by (21) if either $p > 1$ *or* $q > 1$, *then*

$$S_k(x_n) \leq c_{12}, \forall k, \forall n$$

for some constant c_{12}.

7. Summary and Conclusions

We have shown a number of results on the theoretical limitation of using parallelism for solving recurrences. For pth order linear recurrences, with k processors the speed-ups are shown to be bounded by ck + d for some constants c, d, with $c < 1$, *no matter how large the size of the problems*. The sharp upper bound is obtained for first order linear recurrences. For nonlinear recurrences of degree > 1, the speed-ups are shown to be bounded by a constant, *no matter how many processors are used and how large the size of the problems*. This is probably the first and may be the only known example of a nontrivial problem which cannot be essentially sped up. By these results we wish to demonstrate that the gain from parallelism very much depends upon the nature of individual problems, e.g., the dependency relationships among the variables of the problems. We believe that to identify properties which prevent us from getting good speed-ups is fundamental for understanding parallel computation.

References

Barnes [68] Barnes, G. H., et al. The ILLIAC IV computer. *IEEE Trans.Comp. C-17* (1968), 746-757.

Borodin and Munro [75] Borodin, A. and Munro, I., *Computational Complexity of Algebraic Numeric Problems*, American Elsevier, New York, N.Y., 1975.

Brent [70] Brent, R. P. On the addition of binary numbers. *IEEE Trans. Comp. C-19* (1970), 758-759.

Brent [74] Brent, R. P. The parallel evaluation of general arithmetic expressions. *JACM 21* (1974), 201-206

Chen and Kuck [75] Chen, S. C. and Kuck, D. J. Time and parallel processor bounds for linear recurrence systems. *IEEE Trans. Comp. C-24* (1975), 701-717.

Heller [74a] Heller, D. A determinant theorem with applications to parallel algorithms. *SIAM J. Numer. Anal. 11* (1974), 559-568.

Heller [74b] Heller, D. On the efficient computation of recurrence relations. Technical report, ICASE, NASA Langley Research Center, Hampton, Va., 1974. Technical report. Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., 1974.

Heller [75] Heller, D. A survey of parallel algorithms in numerical linear algebra. Technical report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., 1975.

Hyafil and Kung [74] Hyafil, L. and Kung, H. T. Parallel algorithms for solving triangular linear systems with small parallelisms. Contributed paper, *2nd Langley Conference on Scientific Computing*, Virginia Beach, Va., 1974. Technical report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., 1974.

Hyafil and Kung [75a] Hyafil, L. and Kung, H. T. The complexity of parallel evaluation of linear recurrences. *Proceedings 7th Annual ACM Symposium on Theory of Computing*. Albuquerque, N.M., 1975, 12-22. To appear in *JACM*.

Hyafil and Kung [75b] Hyafil, L. and Kung, H. T. Parallel evaluation of recurrences and parallel algorithms for solving band linear systems. Technical report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa. 1975.

Kogge [74] Kogge, P. M. Parallel solution of recurrence problems. *IBM J. R and D 18* (1974), 138-148.

Kogge and Stone [73] Kogge, P. M. and Stone, H. S. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comp. C-22* (1973), 786-793.

Kuck and Maruyama [73] Kuck, D. J. and Maruyama, K. M. The parallel evaluation of arithmetic expressions of special forms. Technical report, RC-4276, IBM Research Center, Yorktown Heights, N.Y., 1973.

Kuck and Maruyama [75] Kuck, D. J. and Maruyama, K. M. Time bounds on parallel evaluation of arithmetic expressions. *SIAM J. Computing 4* (1974), 147-162.

Kung [74] Kung, H. T. New algorithms and lower bounds for the parallel evaluation of certain rational expressions. *Proceedings 6th Annual ACM Symposium on Theory of Computing*, Seattle, Wa., 1974, 323-333. To appear in *JACM*.

Lambiotte and Voigt [74] Lambiotte, J. J., Jr. and Voigt, R. G. The solution of tridiagonal linear systems on the CDC STAR-100 computer. Technical report, ICASE, NASA Langley Research Center, Hampton, Va., 1974.

Munro and Paterson [73] Munro, I. and Paterson, M. Optimal algorithms for parallel polynomial evaluations. *JCSS 7* (1973). 189-198.

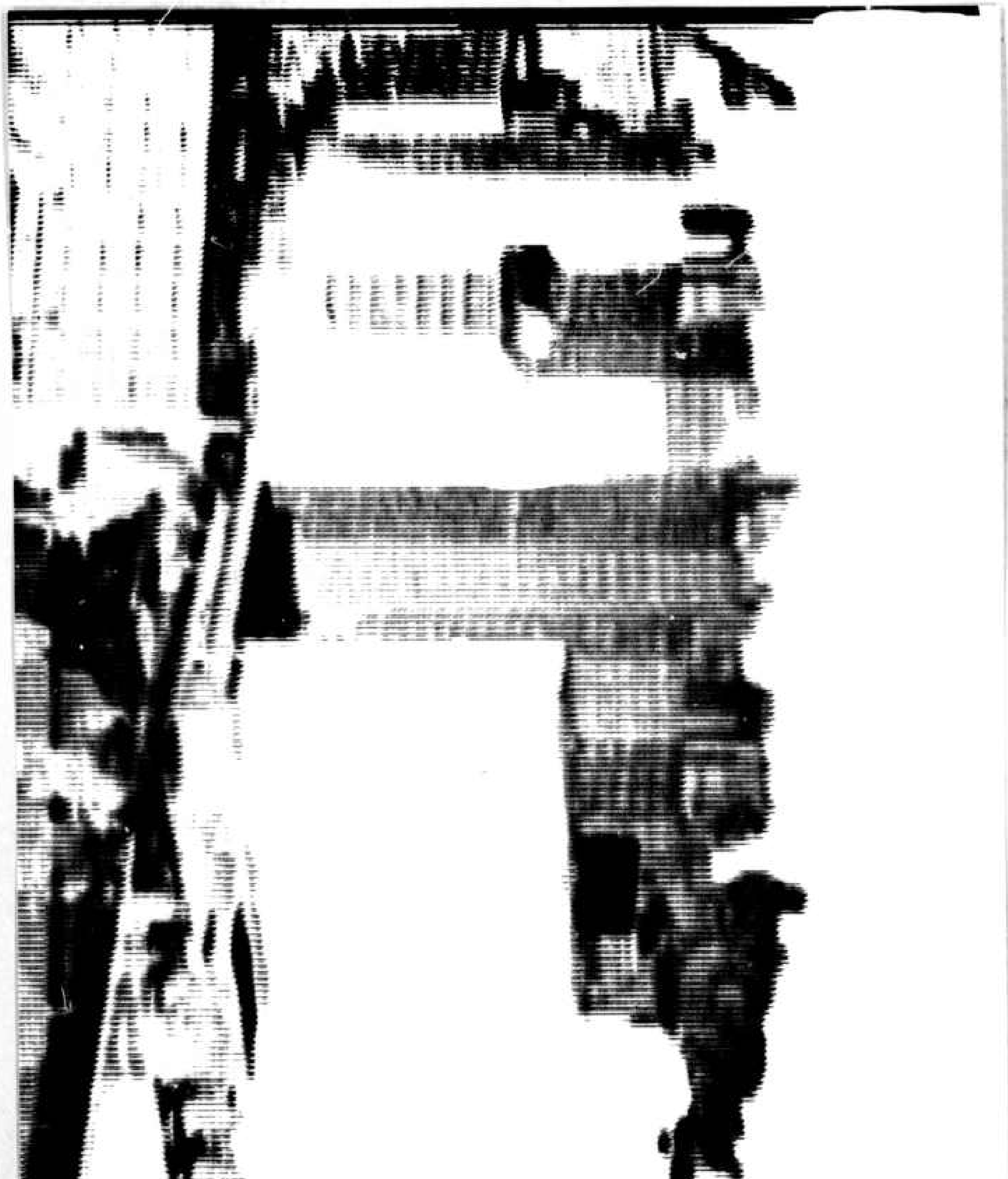
Orcutt [74] Orcutt, S. E. Parallel solution methods for triangular linear systems of equations. Technical report, 77, Digital Systems Lab., Stanford University, Stanford, Ca., 1974.

Stone [73] Stone, H. S. An efficient parallel algorithm for the solution of a tridiagonal system of equations. *JACM 20* (1973), 27-38.

Stone [74] Stone, H. S. Parallel tridiagonal solvers. Technical report, Digital Systems Lab., Stanford University, Stanford, Ca., 1974.

Strassen [69] Strassen, V. Gaussian elimination is not optimal. *Numerische Mathematik 13* (1969), 354-356.

Winograd [74] Winograd, S. On the parallel evaluation of certain arithmetic expressions. Technical report, RC-4808, IBM Research Center, Yorktown Heights, N.Y., 1974. To appear in *JACM*.



Overview of the Hearsay Speech Understanding Research

Lee D. Erman

Hearsay is the generic name for much of the speech understanding research in the computer science department at Carnegie-Mellon University (CMU). The major goals of this research include the investigation of computer knowledge-based problem-solving systems and the practical implementation of speech input to computers. An emphasis of this effort is the design of system structures for efficient implementation of such systems.

We will first describe the problem of speech understanding and (in Section 2) present the context of the Hearsay effort. Section 3 describes the Hearsay model and implementation philosophy. Then, in Section 4, HearsayI is described, including some major design limitations which formed much of the motivation for HearsayII, described in Section 5.

I. The Problem of Speech Understanding

In order to provide a framework for discussion, a conceptual model of speech communication is presented:

1) The purpose of a speech utterance is to transmit information from the speaker to the listener.
2) The speaker starts with some deep semantic representation of the message. Several kinds of transformations are applied to this representation (syntactic, linguistic, phonological, neurological, articulatory, acoustic, etc.). The result is an acoustic signal.
3) The acoustic signal is detected by the listener. The listener applies transformations which are similar (though inverse) to those of the speaker; the result is some semantic representation for the listener.
4) The correctness or effectiveness of the transmission is related to the correspondence between the meaning that the speaker intends and the meaning derived by the listener; it is measured in behavioral terms—i.e., what actions of the listener are triggered by receipt of the message. (Very often this behavior takes the form of an utterance generated by the original listener.)

The goal of automatic speech understanding is to produce a machine (usually in the form of a computer program) which can effectively perform as the listener.

The problem of understanding speech with the competence of a human is formidable. A reasonable plan is to approach the most general kinds of solutions by designing and building a sequence of systems, each of which is more ambitious than the previous. There are many dimensions along which to move to provide this graded sequence (e.g., requirements of vocabulary size, speed of response, accuracy, number of speakers). A way of capturing these various dimensions is the concept of a *task*—a well-defined domain within which the machine is to perform some functions. For example, the task might be to answer the user's (speaker's) questions about airline flight schedules or to provide an interactive computer-programming facility. In defining a task, one important aspect is the spoken *input language*. This language is pre-specified lexically, syntactically, and semantically; that is, descriptions are given of the words, how they may be sequenced to form sentences, and the meaning of the sentences in the context of the task.[1]

There are two major aspects of the speech communication process which generate most of the problems in machine understanding:

1) *The nature of the speech signal*—The transformations involved in speech production are many and complex, and they strongly interact with each other. The result is a very large amount of *variability* in the signal which conveys little or no meaning, i.e., which is noise in the context of the speech understanding task. Repetitions of the "same" utterance, spoken by one speaker under unchanging conditions just seconds apart often

1 This use of a task to constrain the problem is not as artificial as it may first appear. Usually human speech understanding is also performed in "constrained domains"—in almost any given situation only a small subset of all possible messages is likely.

result in significant variation of the signal. As the various conditions (e.g., identity, age, gender, emotional state, and environment of the speaker) are relaxed and allowed to change, this variability increases significantly. Further, strong interactions occur among the various elements; that is, words, phones, phrases, etc., influence and modify nearby words, phones, phrases, etc., and thus have differing manifestations in different contexts.[2]

2) *The nature of our knowledge of the transformations*—Theories which attempt to explain the production of speech are, in general, incomplete and inadequate in explaining the phenomena with a great deal of accuracy. Also, it is often difficult to translate existing theories into the framework of feasible recognition algorithms.

Largely because of these two aspects, the kinds of machine speech understanding systems developed can be characterized as having several interesting and problem-laden features:

1) The system must make use of multiple and diverse sources of *knowledge* to solve the problem (e.g., acoustic-phonetics, phonology, syntax, semantics, pragmatics); these knowledge sources (*KS*s) correspond to the different kinds of transformations that generate the speech signal. Designing an effective control structure for these many diverse KSs is crucial and difficult.

2) Each source of knowledge is *incomplete and errorful*. Thus, although it is used in an attempt to further the recognition of an utterance, each KS will also introduce errors into the analysis process. The different sources must work to correct each other's mistakes in order to keep errors from propagating excessively.

3) The systems developed tend to be *large and complex*. Building, debugging, understanding, and evaluating them is difficult. In particular, many researchers need to interact with the system over a period of several years, both experimenting with its operation and modifying it. An important aspect of system modification is the ability to modify and replace individual KSs.

4) Because of the effectiveness and apparent ease of human performance in the speech understanding task, a useful solution to the problem must be a system which approaches that *performance*, primarily in terms of speed, accuracy, and, ultimately, economy.

5) Because the systems tend to be highly experimental, they must be exercised often and over substantial amounts of trial data. The performance of the system *while under development* (particularly in terms of speed of execution) is an important factor in determining how much experimentation can occur. Thus, issues of performance are crucial even in the development stage.

6) Because the systems are complex and experimental, the interface through which the researcher controls and interacts with the system is crucial. The researcher must be able to interact with the system flexibly and at the functional level of the system (in addition to the more traditional machine language and programming language levels).

This has been a short introduction into the problems of developing speech understanding systems. A more complete analysis of the problem, including pointers to the relevant literature, can be found in Newell et al. [71].

2. Context of This Work

Hearsay's direct lineage can be traced back ten years. The work of Reddy and Reddy & Vicens at Stanford University (Reddy [66]; Reddy and Vicens [68]; Vicens [69]) resulted in extending the state-of-the-art of isolated word recognition systems (e.g., 91% accuracy on a 561-word lexicon in ten times real-time on a PDP10 and with live input). This system differed from most earlier ones, which were essentially pattern classifiers, in that it contained a substantial amount of speech knowledge and it used extensive heuristics in applying the knowledge to prune the search space. In addition, one version of the system was created which used syntactic constraints and operated on connected speech (although in a very *ad hoc* and unextendable manner).

The Hearsay model for speech understanding was developed at CMU during 1970-1971 (Reddy, Erman, and Neely [70]; Reddy [71]; Reddy, Erman, and Neely [72]). This model faced the problems of speech understanding (i.e., in a task domain) and connected speech. The Hearsayl system was designed and built as an implementation of this model (Reddy, Erman, and Neely [73]; Reddy, Erman, Fennell, and Neely [73]; Neely [73]; Erman [74]). This system, which was the first demonstrable live system to handle non-trivial connected speech, became operational in June, 1972, and has been since augmented and studied (Lowerre [75]). Although a number of simplifying assumptions were made in implementing the model, Hearsayl does address the problems of connected speech and of the role and interactions of different kinds of knowledge. By exhibiting a successfully working system which is

2 We are concerned here with *connected speech* input, as opposed to *isolated word* systems in which the words (for short phrases treated as indivisible units) are spoken individually.

based on a model and by providing a set of solutions to these problems (even if some of the solutions are known to be far from optimal), Hearsayl clarified the problems and serves as a basis, and encouragement, for subsequent work.

The experience of building and experimenting with Hearsayl, together with the other research in the field, led to a design review which resulted in the Hearsayll system (Lesser, Fennell, Erman, and Reddy [74]). Hearsayll is also based on the Hearsay model; it generalizes and extends many of the concepts which exist in a more simplified form in the Hearsayl system.

Concurrent with the early stages of the Hearsay development, a group was formed by the Advanced Research Projects Agency (ARPA) to study the feasibility of developing speech understanding systems. This group, which included researchers active in artificial intelligence as well as those in more traditional directions of speech recognition research, produced its report in May, 1971. This report (Newell et al. [71]) provides a comprehensive and detailed analysis of the problems involved. Part of this study included the specification of a set of nineteen dimensions for describing the capabilities of a speech understanding system—the first column of Figure 1 summarizes those dimensions.

On recommendation of the study group, a five-year ARPA Speech Understanding Research effort was launched in October, 1971. An innovative plan with five principal contractors (including CMU) was chosen; each was to aim to produce a complete system meeting a set of specifications laid out by the study group (the second column of Figure 1) and all were to interact, exchanging ideas and data. Although charged to meet the same set of specifications, each group was free to choose its own orientation (and task domains). Thus, the flavor of each of the systems reflects the particular expertises and motivations of the people involved.

The Hearsay research represents CMU's major efforts to meet the ARPA specifications; in particular, it is hoped that Hearsayll will accomplish that goal. In addition, several other systems are being experimented with, also aiming to meet these goals: a version of the Dragon system (Baker [75]) being extended by Reddy and Lowerre and a combination of Hearsayl and Dragon (Lowerre [75]).

In this paper we will describe only the Hearsay effort. An IEEE symposium on speech recognition was held at CMU in April, 1974, at which most workers in the field were represented. The contributed and invited papers from that symposium (Erman [74b]; Reddy [75]) provide a comprehensive description of the state-of-the-art at that time.

3. The Hearsay Model and Implementation Philosophy

This section describes a general model of speech understanding, the "Hearsay model", and some of the problems implied by that model. The following two sections provide overviews of the Hearsayl and Hearsayll implementations of that model.

As one knowledge source (KS) makes errors and creates ambiguities, other KSs must be brought to bear to correct and clarify those actions. This KS cooperation should occur as soon as possible after the introduction of an error or ambiguity in order to limit its ramifications. The mechanism used for providing this high degree of cooperation is the *hypothesize-and-test* paradigm. In this paradigm, solution-finding is viewed as an iterative process. Two kinds of KS actions occur: a) the creation of an *hypothesis*, an "educated guess" about some aspect of the problem, and b) tests of the plausibility of the hypothesis. For both of these steps, the KS uses a *priori* knowledge about the problem, as well as the previously generated hypotheses. This "iterative guess-building" terminates when a consistent subset of hypotheses is generated which satisfies some specified requirements for an overall solution.

As a strategy for developing such systems, one needs the ability to add and replace KSs and to explore different control strategies. Thus, such changes must be relatively easy to accomplish; there must also be ways to evaluate the performance of the system in general and the roles of the various KSs and control strategies in particular. This ability to experiment conveniently with the system is crucial if the amount of knowledge is large and many people are needed to introduce and validate it. One means of helping to provide these flexibilities is to require that KSs be independent; i.e., the explicit interactions between KSs and their assumptions about each other must be minimal.

Besides providing for the modification and evaluation of KSs, decomposition of the system into relatively independent KSs also facilitates its implementation on an asynchronous multi-processor machine. Such configurations seem increasingly attractive as cost-effective ways of obtaining large amounts of computing power. One problem that has limited the development and usage of such machines is the difficulty of decomposing large problems for such machines. Erman, Fennell, Lesser, and Reddy [73] describe this problem and outline some early solutions in the Hearsay context; Lesser [75] provides a survey of this subject.

The basic view of development of a speech understanding system includes a strong component of experimentation: one needs to build a system and

Dimensions and Examples	ARPA Specifications for 1976 Systems
	The system should:
(1) *Manner of Speech* connected? isolated words?	(1) accept connected speech
(2) *Number of Speakers* one? small set? open population?	(2) from many
(3) *Dialect and Manner* cooperative? casual? single gender? both genders? children? what dialect(s)?	(3) cooperative speakers of the "general American dialect",
(4) *Environmental Conditions* quiet room? computer room? factory? public place?	(4) in a quiet room
(5) *Transducer* high quality microphone? telephone?	(5) over a good quality microphone
(6) *Speaker Tuning* few sentences? paragraphs? full vocabulary?	(6) allowing slight tuning of the system per speaker,
(7) *Speaker Training* natural adaptation? elaborate?	(7) but requiring only natural adaptation by the user,
(8) *Vocabulary Size and Selection* 50? 200? 1,000? 10,000? preselected? selective rejection? free?	(8) permitting a slightly selected vocabulary of 1,000 words,
(9) *Grammar* fixed phrases? artificial language? free English? adaptable?	(9) with a highly artificial syntax,
(10) *Task* highly constrained (e.g., simple retrieval)? focussed (e.g., numerical algorithms)? open?	(10) and a task with a constrained and fairly simple semantics,
(11) *User Model* nothing? current knowledge about the user?	(11) with a simple psychological model of the user,
(12) *Mode of Interaction* response only? ask for repetitions? explain language? discuss communications?	(12) providing graceful interaction,
(13) *Error Rate* none (<0.1%)? <10%? >20%?	(13) tolerating less than 10% semantic error,
(14) *Response Time* no hurry? few times real-time? immediate?	(14) in a few times real-time,
(15) *Processing Power* 1×10^8 instructions/sec? 10 mips? 100 mips? 1000 mips?	(15)
(16) *Memory Size* 1 megabit? 10mb? 100mb? 1000mb?	(16)
(17) *System Organization* simple program? multiprocessing? parallel processing? unidirectional processing? feedback? backtrack? planning?	(17)
(18) *Cost* $0.001/sec. of speech? $0.01/s? $0.1/s? $1.0/s?	(18)
(19) *Operational Date*	(19) and have a prototype demonstrable in 1976.

Figure 1: Dimensions of Speech Understanding Systems and ARPA Specifications for 1976. (After Newell et al. [71].)

then manipulate and study it. In order to provide an environment to accomplish this, a two-level approach is taken: First, a basic set of facilities is provided, and, second, various configurations are built using these facilities. These facilities, which together are called the *kernel*, form a problem-dependent programming system for building and experimenting with particular configurations. The correct choice of kernel facilities and their implementation are crucial ingredients in developing a system.

The Blackboard—Representation of Knowledge

The requirement that KSs be independent implies that the functioning (and very existence) of each must not be necessary or crucial to the others. On the other hand, the KSs are required to cooperate in the iterative guess-building, using and correcting one another's guesses; this implies that there must be interaction among the processes. These two opposing requirements have led to a design in which each KS interfaces to the others externally in a uniform way that is identical across KSs and in which no knowledge source knows what or how many other KSs exist. The interface is implemented as a dynamic global data structure, called the *blackboard*.[3] The primary units in the blackboard are the guesses about particular aspects of the problem—the hypotheses. At any time, the blackboard holds the current state of the system; it contains all the guesses about the problem that exist. Subsets of hypotheses represent partial solutions to the entire problem; these may compete with the partial solutions represented by other (perhaps overlapping) subsets.

Each KS may access information in the blackboard. Each may add information to the blackboard by creating (or deleting) hypotheses, by modifying existing hypotheses, and by establishing or modifying explicit structural relationships among hypotheses. The generation and modification of globally accessible hypotheses is the exclusive means of communication among the diverse KSs. This mechanism of cooperation, which is an implementation of the hypothesize-and-test paradigm, allows a KS to contribute knowledge without being aware of which other KSs will use the information or which KS supplied the information that it used. It is in this way that KSs are made independent and separable. The structural relationships form a network of the hypotheses and are used to represent the deductions and inferences which caused a KS to generate one hypothesis from others. The explicit retention in the blackboard of these dependency relationships is used to hold, among other things, competing hypotheses.

Because of the central importance of the blackboard, its design (i.e., the design of the structure of hypotheses and their relationships) is crucial. This is usually called the problem of *representation*.

Activation of Knowledge Sources—Focus of Attention

An action of a KS in the blackboard takes place in the context of some hypotheses already existing in the blackboard. For example, a KS which hypothesizes words may require a stressed vowel (as well as some surrounding sounds) as its context in order to consider generating new word hypotheses.

At any time there may be many different contexts which satisfy the needs of one or more KSs. The problem of choosing the order for activating KSs on contexts is generally called the problem of *control flow*. Because there may be many such possible activations and because each activation of a KS will, in general, create the potential for even more activations (e.g., the word hypothesizer, given a single new stressed vowel context, might hypothesize five new words as competing candidates—each of these might provide a new context for a syntactic parser), the number of possible activations may grow.

If very, very large amounts of processing power (and memory) were available, one could consider actually activating all KSs in all their possible contexts. This would expand the blackboard with many (competing) hypotheses. Assuming this would eventually terminate (i.e., at some point no new contexts are created), a decision process could then try to pick from all the competing hypotheses that subset which best describes the data—this would be the system's "solution" to the problem. Because of this combinatoric explosion of possibilities (caused mostly by the problems of variability and incompleteness in the signal and errorfulness of the KSs), this complete expansion is not feasible. Therefore, the control strategy can pick only a small subset of the applicable KS activations; this can be thought of as exploring a limited portion of the (potential) fully-expanded blackboard. The problem of choosing a control strategy which can efficiently reach the correct set of hypotheses is called the *attention-focusing* problem. Its solution is also critical for the success of a system: If portions of the correct solution are pruned, the solution will never

3 The term "blackboard" was used by Simon [66] in describing a mechanism in long-term memory as part of a theory of the psychology of problem-solving. Simon [71] further develops this concept and elaborates its uses in the context of an abstract model for problem-solving.

be found; if many incorrect portions are not pruned, the combinatoric explosion will use large amounts of computing resources (and may also force the system to give up before reaching the solution).

The problems of representation of knowledge and searching a large solution space (focus of attention) are two of the central problems of artificial intelligence. The speech understanding problem, with its requirements of high performance and the use of diverse and errorful KSs, provides a rich field for their study.

4. Overview of Hearsayl

The blackboard of Hearsayl consists of partial sentence hypotheses, each of which is a sequence of words with non-overlapping time locations in the utterance. Each is a *partial* sentence hypothesis because not all of the utterance need be described by the given sequence of words. In particular, gaps of one or more words of the utterance which have not yet been hypothesized (in the context of the particular sentence hypothesis) are designated by "filler" words. The partial sentence hypotheses also contain confidence ratings for each word hypothesis and a composite rating for the overall sequence of words. A sentence hypothesis is the focal point that is used to invoke a KS. The sentence hypothesis also contains the accumulation of all information that all KSs have contributed to that hypothesis.

System activity goes through a number of cycles. In each cycle there is one partial sentence hypothesis on the blackboard which is the focal point of activity; this focal hypothesis forms the context for KS activity during the cycle. KSs are activated in a lockstep sequence consisting of three phrases per cycle: *poll*, *hypothesize*, and *test*. At each phase, all KSs are activated for that phase, and the next phase does not commence until all KSs have completed the current one. The *poll* phase involves determining which KSs have something to contribute to the focal sentence hypothesis; polling also determines how confident each KS is about its proposed contributions. The *hypothesize* phase consists of activating the KS showing the most confidence about its proposed contribution of information. This KS then hypothesizes a set of possible words (option words) for some (one) "filler" word in the speech utterance. The *testing* phase consists of each KS evaluating (verifying) the possible option words with respect to the given context. After all KSs have completed their verifications, the option words which seem most likely, based on the combined ratings of all the KSs, are then used to construct new partial sentence hypotheses. The blackboard is then re-evaluated to find the most promising sentence hypothesis; this hypothesis then becomes the focal point for the next hypothesize-and-test cycle.

A mediator module (the "recognition overlord") is responsible for maintaining the blackboard, calculating combined ratings from the ratings assigned to hypotheses by the individual KSs, and deciding when to stop and accept a solution (or give up). The rating of the sentence hypotheses is the mechanism for attention focusing. A best-first strategy is used—the currently highest rated hypothesis is the one used as the context for the next cycle. If an error is made, the rating of the incorrect hypothesis will, hopefully, eventually degrade and attention will be focused to the sentence hypothesis which now has the highest rating.

Hearsayl contains three KSs:

1) The ac*oustic-phonetic* KS deals with the sounds of the words of the input language and how they relate to the speech signal. It obtains (from a preprocessing module called *EAR*) a representation of the speech signal as an errorful (or course!) sequence of segments, each segment being labeled with a phonetic-like label. The input language is specified to the KS as a lexicon of words in which each word is "spelled" as a sequence of phonemic symbols (with some alternative spellings). The KS both hypothesizes words (from the segments) and evaluates the word hypotheses of other KSs.

2) The *syntax* KS deals with the orderings of words in the utterance according to the specified grammar of the input language. This grammar is specified to the KS in BNF notation. Given some contiguous word hypotheses, the KS can evaluate them for consistency with the grammar and also can hypothesize additional words which are likely to occur contiguous to them.

3) The *semantics* KS deals with the meaning of words and phrases of the input language, in the context of the task. Only one task semantics KS has been programmed (for "Voice-chess"—playing a game of chess verbally); its design is highly explicit to the one task. This KS hypothesizes and rates sentences and portions of sentences based on the chess moves they represent; it uses both the legality of the move in the current chess board position as well as the "goodness" of the move (as determined by a chess-playing program which the KS consults).

Hearsayl Performance

The Hearsayl system first demonstrated live, connected-speech recognition in June, 1972, at a workshop held at CMU. Since that time, about two person-years have been spent in studying it and in

improving its performance. The system has been formally tested on a set of 144 connected speech utterances, containing 676 word tokens, spoken by five speakers, and consisting of four tasks (only one of which has had a semantics component programmed), with vocabularies ranging from 28 to 76 words. The system locates and correctly identifies about 93% of the words, using all three of its KSs. Without the use of the semantics KS, the accuracy decreases to 70%. It decreases further to about 30% when neither syntax nor semantics are used. Hearsayl operates in about 7 to 10 times real-time on a PDP10-KA10 (0.3 million instructions/sec. machine), using about 120K words (36-bits/word) for storage and programs.

Hearsayl Design Limitations

There are four major design decisions in the Hearsayl implementation of knowledge representation and cooperation which make it difficult to directly extend Hearsayl to more ambitious performance goals.

The *first*, and most important, of these limiting decisions concerns the use of the hypothesize-and-test paradigm. As implemented in Hearsayl, the paradigm is exploited only at the word level. That is, the information content of any hypothesis in the blackboard is limited to a description at the word level. The addition of non-word level KSs (i.e., KSs cooperating via either sub-word levels, such as syllables or phones, or via supra-word levels, such as phrases or concepts) thus becomes cumbersome because this knowledge must somehow be related to hypothesizing and testing at the word level.

Secondly, Hearsayl constrains the hypothesize-and-test paradigm to operate in a lockstep control sequence. The effect of this decision is to limit parallelism of execution (and thus reduce effectiveness on a multi-processor configuration); this is because the time required to complete a hypothesize-and-test cycle is the maximum time required by any single hypothesizer KS plus the maximum time required by any single verifier (testing) KS. Another disadvantage of this control scheme is that the time increases for the system to refocus attention, because there is no provision for any communication of partial results among KSs. Thus, for example, a rejection of a particular option word by a KS will not be noticed until all the KSs have tested all the option words.

A *third* weakness in the Hearsayl implementation concerns the structure of the blackboard: there is *no* provision for specifying relationships among alternative sentence hypotheses. This absence has the effect of increasing the overall computation time and increasing the time to refocus attention, because the information gained by working on one hypothesis cannot be shared by propagating it to other relevant hypotheses.

A *fourth* limiting design decision relates to how a global problem-solving strategy is implemented in Hearsayl: The policies for attention-focusing and control are embedded in the recognition overlord module in an *ad hoc* fashion—there is no coherent structure for the algorithms and they are "wired in" to the kernel of the system, rather than being available for easy manipulation and experimentation. Thus it is awkward to modify and evaluate policy algorithms.

5. Overview of HearsayII

HearsayII represents the step following Hearsayl in the sequence of increasingly ambitious systems for speech understanding. The major changes to the system structure are a) in the representation of knowledge in the blackboard and b) in the manner of activation and attention-focusing of KSs.

The Blackboard of HearsayII

The blackboard has been extended and generalized to allow a) the representation of all levels of information (acoustic, phonetic, syllabic, etc.) in addition to the lexical and sentence levels of Hearsayl and b) the explicit representation of relationships among hypotheses.

The blackboard is partitioned into distinct information *levels*; each level is used to hold a different (and potentially complete) representation of the utterance. Associated with each level is a set of primitive elements appropriate for representing the problem at that level. (For example, the elements at the lexical level are the *words* of the vocabulary to be recognized, while the elements at the phonetic level are the *phones* of English.) Each hypothesis exists at a particular level and is labeled as being a particular element of the set of primitive elements at that level. The choice of levels (and the set of elements at each level) is not prespecified by the kernel of the system. To the kernel, all levels are uniform; so new ones can be added at any time. The configuration of levels that is currently in use is shown in Figure 2.[4]

Parametric Level—The parametric level holds the most basic representation of the utterance that the system has; it is the only direct input to the machine about the acoustic signal. Several different sets of parameters are being used in HearsayII interchangeably: 1/3-octave filter-band energies measured every 10 msec., LPC-derived vocal-tract parameters, and wide-band energies and zero-crossing counts.

4 An elaboration of the following description can be found in Shockey and Erman [74].

Segmental Level—This level represents the utterance as labeled acoustic segments. Although the set of labels is phonetic-like, the level is not intended to be phonetic—the segmentation and labeling reflect acoustic manifestation and do not, for example, attempt to compensate for the context of the segments or attempt to combine acoustically dissimilar segments into (phonetic) units.

Phonetic Level—At this level, the utterance is represented by a phonetic description. This is a *broad* phonetic description in that the size (duration) of the units is on the order of the "size" of phonemes; it is a *fine* phonetic description to the extent that each element is labeled with a fairly detailed allophonic classification (e.g., "stressed, nasalized [I]").

Surface-Phonemic Level—This level, named by seemingly contradicting terms, represents the utterance by phoneme-like units, with the addition of modifiers, such as stress and boundary (word, morpheme, syllable) markings.

Syllabic Level—The unit of representation here is the syllable.

Lexical Level—The unit of information at this level is the word.

Phrasal Level—Phrases appear at this level. In fact, since a level may contain arbitrarily many "sub-levels" of elements (using "links", as described below), traditional kinds of syntactic trees are directly represented here.

The decomposition of the blackboard into distinct levels of representation can also be thought of as an a priori framework of a *plan* for problem-solving. Each level is a generic stage in the plan. The goal at each level is to create and validate hypotheses at that level. For example, the goal at the phonetic level is a phonetic transcription of the utterance. The overall goal of the system is to create (using "links", as described below) the most plausible network of hypotheses that sufficiently covers the levels. 'Plausible and sufficient' here refer to the judgment of the KSs; 'covering the levels' means a network that connects hypotheses which describe the speech signal (at the parametric level) to hypotheses which describe the semantic content of the utterance (at the phrasal level).

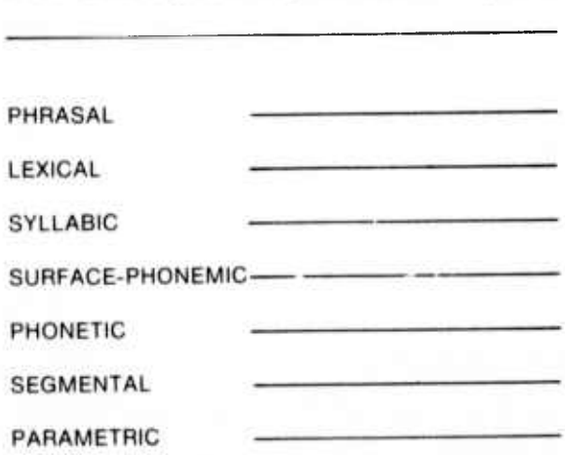


Figure 2: The Levels Currently Used in Hearsayll.

The decomposition of the problem space into more levels than in Hearsayl parallels the desire to decompose the KSs more finely, yielding more KSs, each of which is simpler and smaller. The principal resultant change in the configuration of KSs is that the single acoustic-phonetic KS of Hearsayl is decomposed into about six KSs currently in Hearsayll. For most KSs, the KS needs to deal with only one or two levels to apply its knowledge; it need not even be aware of the existence of other levels. Thus, each KS can be made as simple as its knowledge allows; its interface to the rest of the system is in units and concepts which are natural to it. Also, new levels can be added as new KSs are designed which need to use them. (For example, the syllabic level was a fairly late addition to the configuration—only two KSs needed to be modified when it was added.)

Activation of Knowledge Sources

A KS is instantiated as a knowledge-source process whenever the blackboard exhibits characteristics which satisfy a "precondition" of the KS. A *precondition* of a KS is a description of some partial state of the blackboard which defines when and where the KS can contribute its knowledge by modifying the blackboard. A KS carries out these actions with respect to a particular *context*, the context being some arbitrary subset of the previously generated hypotheses in the blackboard. Thus, new hypotheses or modifications to existing hypotheses are constructed from the (static) knowledge of the KS and the educated guesses made at some previous time by another KSs.

The modifications made by any given KS process are expected to trigger further KSs by creating new conditions in the blackboard to which those KSs, in turn, respond. The structure of a hypothesis is designed to allow the preconditions of most KSs to be sensitive to a single, simple change in some hypothesis (e.g., the creation of a new hypothesis of a particular type, a change of a rating, or the creation of a structural link between particular kinds of hypotheses). Through this *data-directed* interpretation of the hypothesize-and-test paradigm, KSs can

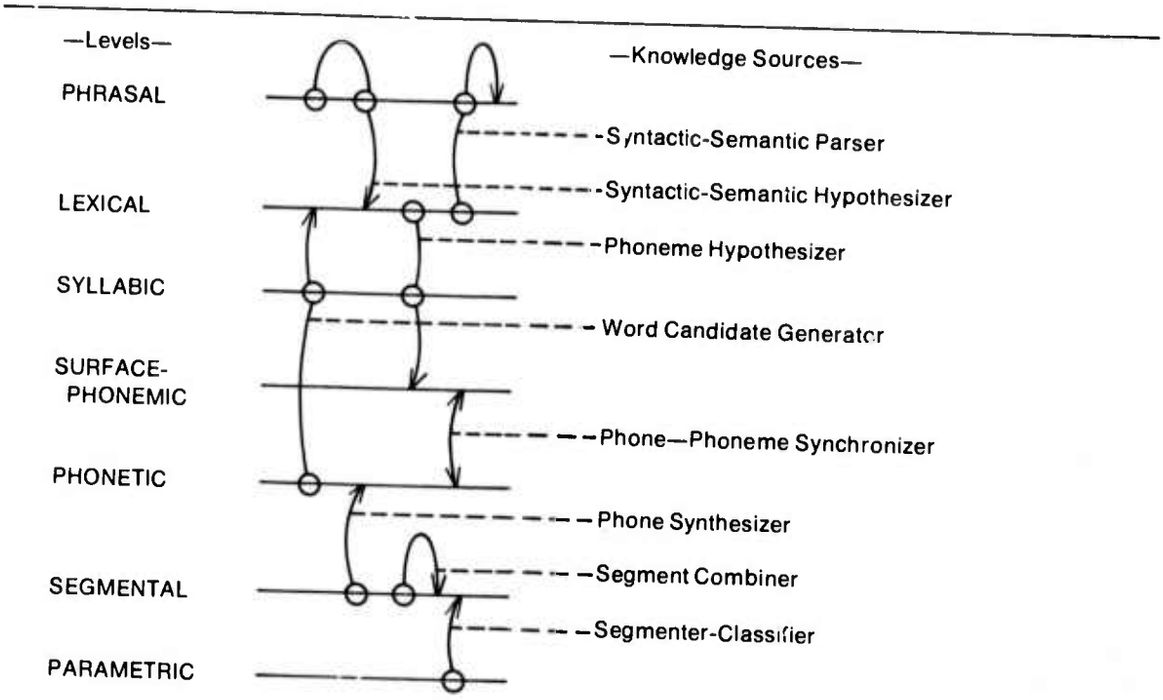


Figure 3: The Current Knowledge Sources in Hearsayll.

also exhibit a high degree of asynchronous activity and potential parallelism.[5]

As examples of KSs, Figure 3 shows many of the current set. The levels are indicated by horizontal lines in the figure and are labeled at the left. The KSs are indicated by arcs connecting levels; the starting point(s) of an arc indicates the level(s) of major "input" for the KS, and the end point indicates the "output" level where the KS's major actions occur. In general, the action of most of these particular KSs is to create links between hypotheses on its input level(s) and: 1) existing hypotheses on its output level, if appropriate ones are already there, or 2) hypotheses that it creates on its output level.

5 One might think of this model for data-directed activation of KSs as a production system (Newell [73]) which is executed asynchronously. The preconditions correspond to the left-hand sides (conditions) of productions, and the KSs correspond to the right-hand sides (actions) of the productions. Conceptually, these left-hand sides are evaluated continuously. When a precondition is satisfied, an instantiation of the corresponding right-hand side of its production is created; this instantiation is executed at some arbitrary subsequent time (perhaps subject to instantiation scheduling constraints).

The *Segmenter-Classifier* KS uses the parametric description of the speech signal to produce a labeled acoustic segmentation. (See Goldberg et al. [75] for a description of the algorithm used.) For any portion of the utterance, several possible alternative segmentations and labels may be produced.

The *Segment Combiner* combines similar adjacent segments into larger units. It is triggered on each new hypothesis at the segmental level.

The *Phone Synthesizer* uses labeled acoustic segments to generate elements at the phonetic level. This procedure is sometimes a fairly direct renaming of an hypothesis at the segmental level, perhaps using the context of adjacent segments. In other cases, phone synthesis requires the combining of several segments (e.g., the generation of [t] from a segment of silence followed by a segment of aspiration) or the insertion of phones not indicated directly by the segmentation (e.g., hypothesizing the existence of an [l] if a vowel seems velarized and there is no [l] in the neighborhood). This KS is triggered whenever a new hypothesis is created at the segmental level.

The *Word Candidate Generator* uses phonetic information (primarily just at stressed locations and other areas of high phonetic reliability) to

generate word hypotheses. This is accomplished in a two-stage process, with a stop at the syllabic level, from which lexical retrieval is more effective. (In fact, there are really two separate KSs here—one that goes from phones to syllables, and one that goes from syllables to words.)

The *Phoneme Hypothesizer* KS is activated whenever a word hypothesis is created (at the lexical level) which is not yet supported by hypotheses at the surface-phonemic level. Its action is to create one or more sequences at the surface-phonemic level which represent alternative pronunciations of the word. (These pronunciations are prespecified as entries in a dictionary.) It also creates the syllable hypotheses for the word, if they do not already exist.

The *Phone—Phoneme Synchronizer* is triggered whenever an hypothesis is created at either the phonetic or the surface-phonemic level. This KS attempts to link up the new hypothesis with hypotheses at the other level. This linking may be many-to-one in either direction.

The *Syntactic-Semantic Parser* uses the syntactic and semantic definition of the input language to build parses at the phrasal level. It is triggered by new word and phrasal hypotheses. This KS is not restricted to left-to-right parsing, but rather works piecemeal wherever hypotheses occur. One of its responsibilities is to identify possible interpretations for the entire utterance. (See Hayes-Roth and Mostow [75].)

The *Syntactic-Semantic Hypothesizer* also uses the syntactic and semantic definition of the input language. It hypothesizes phrasal and word hypotheses which are likely to occur adjacent to phrasal and word hypotheses already on the blackboard. This provides "top-down" activity in the system.

The *Rating Policy* KS operates at all levels of the blackboard. Its function is to propagate evaluations of hypotheses. For each hypothesis, this KS calculates ratings which are based on a) intrinsic ratings placed on the hypothesis by other KSs and b) the hypothesis' relationships to other hypotheses.

Hypotheses: Structure and Interrelationships

As described above, the structure of hypotheses at each level in the blackboard is identical (i.e., the interpretation of hypotheses at different levels is imposed by the KSs dealing with them.) The internal structure of an hypothesis consists of a fixed set of *attributes* (i.e., fields which are named); this set is the same for hypotheses at all levels of representation in the blackboard. The values of the attributes are set and modified by the KSs.

Besides holding information necessary to describe the hypothesis, attributes also serve as mechanisms for implementing the data-directed hypothesize-and-test paradigm. That is, a KS can specify particular attributes of hypotheses (usually at particular levels) which it wants to have monitored; whenever a change is made to one of these monitored attributes, the KS (through its precondition) can be activated and notified of the nature of the change.

Attributes can be grouped into several classes:

- The first class of attributes *names* the hypothesis: It contains the unique name of the hypothesis, the name of its level, and its label from the element set at that level.
- One very important set of attributes specifies *structural relationships* with other hypotheses, as described below.
- The next class of attributes is composed of parameters which *rate* the hypothesis. These include separate numerical ratings derived from a) *a priori* information about the hypothesis (usually placed on the hypothesis by its creator KS), and b) information derived from its relationships to other hypotheses.
- Another set of attributes contains information about KS *attention* to the hypothesis. These include suggestions (by KSs) of what type of further processing should occur. These suggestions are *goals*.
- For speech, *time* is a fundamental concept, so the Hearsayll system has a class of attributes for describing the begin- and end-time and the duration of the event which the hypothesis represents. These attributes include ways of explicitly representing fuzzy notions of the times. Besides its descriptive importance, the time attribute class is used to partition the blackboard for efficient access; e.g., a KS can retrieve hypotheses which overlap a particular time region. Using both time and level, a two-dimensional partitioning occurs.
- The capability for arbitrary *KS-specific* attributes is also included. This can be used by a KS to hold arbitrary information about the hypothesis; in this way a KS need not hold state information about the hypothesis internally across activations of the KS and allows, for example, the implementation of generator functions. If several KSs share knowledge of the name of one of these attributes, each of them can access and modify the attribute's value and thus communicate just as if it were a "standard" attribute;

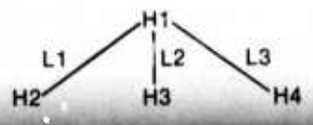
this can be used as an escape mechanism for explicit KS intercommunication.

— A unique class of hypothesis attributes, called *processing state* attributes, contains succinct summaries and classifications of the values of the other attributes. For example, the values of the rating attributes are summarized and the hypothesis is classified as either "unrated", "neutral" (noncommittal), "verified", "guaranteed" (strongly verified and unique), or "rejected". Other processing state attributes summarize the structural relationships with other hypotheses and characterize, for example, whether the hypothesis has been "sufficiently and consistently" described as an abstraction of hypotheses at lower levels. The processing state attributes are especially useful for efficiently triggering KSs; for example, a KS may specify in its precondition that it is to be activated whenever a hypothesis at a particular level becomes "verified". These attributes are also used for the goal-directed scheduling of KSs, as described in the next section.

Given a specific hypothesis, a KS can examine the value of any of its attributes. A KS source also needs the ability to retrieve sets of hypotheses whose attributes satisfy conditions in which the KS is interested; e.g., a KS may want to find all hypotheses at the phonetic level which are vowels and which occur within a particular time range. The system provides an *associative retrieval* search mechanism for accomplishing this. The search condition is specified by a *matching prototype* which is a partial specification of the components of a hypothesis.

Structural relationships between hypotheses in the blackboard are represented through the use of *links*; links provide a means of specifying contextual abstractions about the relationships of hypotheses. A link is an element of the blackboard which associates two hypotheses as an ordered pair; one of the nodes is termed the *upper hypothesis*, and the other is called the *lower hypothesis*. The lower hypothesis is said to *support* the upper hypothesis while the upper hypothesis is called a *use* of the lower one; in general, the lower hypothesis is at the same or a lower level in the blackboard than the upper hypothesis.

There are several types of links, with the types describing various kinds of relationships. Consider this structure:



H1 is the upper hypothesis and H2, H3, and H4 are the lower hypotheses of links L1, L2, and L3, respectively. If the links are all of type *OR*, the interpretation is that H1 is either an H2 or an H3 or an H4. This is one way that alternative descriptions are possible. If the links in the figure are of type *AND*, the interpretation is that all of the lower hypotheses are necessary to support the existence of H1. Variants of the AND- and OR-links are also used. An important one is the *SEQUENCE* link; it is similar to the AND-link except that a contiguous time-ordering is implied on the set of lower hypotheses supporting the upper hypothesis—if the links in the figure are SEQUENCE links, then H4 follows H3 which follows H2.

Besides showing structural relationships between hypotheses (e.g., that one hypothesis is composed of several other units), a link is a statement about the degree to which one hypothesis implies (i.e., "gives evidence for the existence of") another hypothesis. The strength of the implication is held as attributes of the link. The sense of the implication may be negative; that is, a link may indicate that one hypothesis is evidence for the *in*validity of another. This statement of implication may be bidirectional; the existence of the upper hypothesis may give credence to the existence of the lower hypothesis and vice versa. Finally, these relationships can be constructed in an iterative manner; links can be added between existing hypotheses by KSs as they discover new evidence for support.

Just as an hypothesis can have more than one lower link, so it can have several upper links. Each of these represents a different use of the hypothesis; the uses may be competing or complementary. The ability to have multiple uses and supports of the same hypothesis, as opposed to creating duplicates for each competing use and abstraction, serves to keep the blackboard compact and thereby reduces the combinatoric explosion in the search space. Further, since all the information about the hypothesis is localized, all uses and supports of the hypothesis automatically and immediately share any new information added to the hypothesis by any KSs. As changes are made to a hypothesis, some of its uses and supports may conflict with each other; if these conflicts become too large, a KS can decide to resolve them by either eliminating some of the conflicting attributes or by *splitting* the hypothesis into two or more hypotheses, each of which is more internally consistent.

Goal-Directed Scheduling of Knowledge Sources

As described earlier, the overall goal of the system is to create the most plausible network of hypotheses that sufficiently spans the levels. At any

instant of time, the blackboard may contain many incomplete networks, each of which is plausible as far as it goes. Some of these incomplete networks may also share subnetworks. Through KS activity, incomplete networks can be expanded (or contracted) and may be joined together (or fragmented). At any time, there may be many places in the blackboard which satisfy the (precondition) contexts for the activation of particular KSs. The task of *goal-directed scheduling* is that of deciding which of these sites should be allocated computing resources.

Several of the attribute classes of a hypothesis can be helpful in making scheduling decisions. Particularly valuable are the values of the attention attributes, which, as described earlier, are indicators telling how much computation has been expended on the hypothesis and suggestions by KSs of how desirable it is to devote further effort on the hypothesis (along with the kinds of processing that are desirable). The processing-state attributes and the ratings are also valuable for making scheduling decisions.

The implementation of the goal-directed scheduling strategy is separated from the actions of individual KSs. That is, the decision of whether a KS can contribute in a particular context is local to the KS, while the assignment of that KS to one of the many contexts on which it can possibly operate is made more globally. The three aspects of a) decoupling of focusing strategy from KS activity, b) decoupling of the data environment (blackboard) from the control flow (KS activation), and c) the limited context in which a KS operates, together permit a quick refocusing of attention of KSs. The ability to refocus quickly is very important because the errorful nature of the KS activity leads to many incomplete and possibly contradictory hypothesis networks; thus, as soon as possible after a network no longer seems promising, the resources of the system should be employed elsewhere.

Implementation and Current Status

Hearsay II is implemented (as was Hearsayl) on the PDP10 in SAIL (VanLehn [73]), an extended Algol-60. A number of language mechanisms—particularly the flexible macro facility—are used to extend the language to include the kernel of the Hearsayll system; the result is a problem-oriented programming system for writing KSs and exploring various configurations. The major facilities provided include:

- —KS definition facilities,
- —blackboard accessing routines—both direct and associative retrieval,
- —blackboard modification routines,
- —a scheduler which activates KSs,
- —an overlay facility which extends the 256K-word address space so that large configurations can be used,
- —blackboard monitoring and tracing facilities,
- —general-purpose tools for experimenter interaction with KSs, including breakpoints, execution tracing, examination and modification of variables, and execution of functions of the KS,
- —tools for building high-level debugging and interactive features that are KS-specific,
- —a package for graphical output of blackboard structures,
- —a timing package for determining execution costs, and
- —a means of reading "cliche" files—stored sequences of commands used for configuring and controlling the system.

The system that results is highly structured and has many conventions to which the participating researchers must adhere. This is necessary in order to maintain a system that many people are modifying and using concurrently. (There are currently about five people maintaining and modifying the kernel and approximately a dozen others experimenting with various KS configurations—a usable and up-to-date system must be operational at all times.)

The kernel has been operational since spring, 1974, and has gone through several major implementation iterations. All the KSs described above are operational; several of them represent second or third generation versions. Because the overlay facility has only just come up (summer, 1975), performance of the system as a whole is still unknown; the KSs have been developed using small configurations at a time. It is expected that preliminary over-all performance information will be available by the end of 1975, but development will continue over the foreseeable future—as long as progress continues to be made.

Although Hearsayll is running on a uni-processor, it is implemented using multiple processes. The asynchronous nature of KS activation raises a number of issues related to interaction on the blackboard. In particular, because the execution of a KS may be delayed for an arbitrary period following the blackboard modification which triggered the KS, it is possible that intervening actions (of other KSs) may have invalidated its triggering conditions by the time that it actually executes. Mechanisms have been developed to handle these problems. This aspect of the research is described in Lesser, Fennell, Erman, and Reddy [74], Fennell [75], and Fennell and Lesser [75].

The Hearsayll system also contains facilities for

simulating its execution on a multi-processor machine. Here the issues of process interference, resource locking (and process deadlock), and processor utilization are met. The papers referenced in the preceding paragraph also describe these aspects in detail. The simulations, using just a subset of the current KSs[6], indicate that Hearsayll can effectively utilize as many as twelve processors, with even more likely as the other KSs are added and as the scheduler is improved to reduce conflicts.

A preliminary implementation of the Hearsayll kernel has been carried out on C.mmp (CMU's multi-mini-processor). This has validated the multi-processing design of the system. This implementation has been accomplished using the L* system (Newell and Robertson [75]). Much of the further investigation of Hearsayll will take place in this context.

Acknowledgments

A significant portion of the CMU Computer Science Department has been involved in the Hearsay efforts—the author is only one of many.

- —Raj Reddy is responsible for a large measure of the ideas, energy, and vision of this work; without him, the project would not have existed.
- —Richard Fennell, Rick Hayes-Roth, Victor Lesser, Richard Neely, and Linda Shockey have been instrumental in the ideas and their exploration.
- —Allen Newell has provided guidance and encouragement.
- —Greg Gill deserves special mention for his outstanding contribution of programming the Hearsayll kernel (several times).
- —Without taking all the space needed to describe each of their contributions, we would like to acknowledge the efforts of all members of the CMU "speech group", as well as the entire computer science department, for contributing to a first-rate research environment.

I wish to thank Vic Lesser for considerable help with this paper.

6 Only a subset of five KSs was used for these simulations because a) the overhead of simulation is very high and b) when the simulations began many of the current KSs either did not exist or were too undeveloped to use.

References

Baker [75] Baker, J. K. Stochastic modeling as a means of automatic speech recognition. Doctoral Dissertation, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1975.

Erman, Fennell, Lesser and Reddy [73] Erman, L. D., R. D Fennell, V. R. Lesser, and D. R. Reddy. System organizations for speech understanding: Implications of network and multiprocessor computer architectures for AI. *Proc. 3rd Inter. Joint Conf. on Artificial Intel.*, Stanford, CA, 1973, 194-199.

Erman [74] Erman, L. D. An environment and system for machine understanding of connected speech. Doctoral Dissertation, Computer Science Dept., Stanford University; Technical Report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1974.

Erman [74b] Erman, L. D. (Ed.) *Contributed Papers of the IEEE Symposium on Speech Recognition.* April 15-19, 1974, Pittsburgh, Pa., IEEE Cat. No. 74CH0878-9AE. Many of these papers have been reprinted in a special issue of *IEEE Trans. on Acoustics, Speech, and Signal Processing*, ASSP-23, 1 (1975).

Fennell [75] Fennell, R. D. Multiprocess software architecture for AI problem solving. Doctoral Dissertation, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1975.

Fennell and Lesser [75] Fennell, R. D. and V. R. Lesser. Parallelism in AI problem solving: A case study of Hearsayll. Sagamore (NY) Computer Conf. on Parallel Processing, 1975.

Goldberg et al. [74] Goldberg, H. G., D. R. Reddy, and R. Suslick. Parameter independent machine segmentation and labeling. In Erman [74b], 106-111.

Hayes-Roth and Mostow [75] Hayes-Roth, F. and D. J. Mostow. An automatically compilable recognition network for structured patterns. *Proc. 4th Inter. Joint Conf. on Artificial Intel.*, Tbilisi, USSR, 1975.

Lesser, Fennell, Erman, and Reddy [74] Lesser, V. R., R. D. Fennell, L. D. Erman, and D. R. Reddy. Organization of the Hearsayll speech understanding system. In Erman [74b], 11-21. Also appeared in *IEEE Trans. on Acoustics, Speech, and Signal Processing*, ASSP-23, 1, (1975), 11-23.

Lesser [75] Lesser, V. R. Parallel processing in speech understanding systems: A survey of design problems. In Reddy [75], 481-499.

Lowerre [75] Lowerre, B. T. Doctoral Dissertation (in preparation). Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA 1975.

Neely [73] Neely, R. B. On the use of syntax and semantics in a speech understanding system. Doctoral Dissertation, Stanford University; Technical Report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1973.

Newell et al. [71] Newell, A., J. Barnett, J. Forgie, C. Green, D. Klatt, J. C. R. Licklider, J. Munson, R. Reddy, and W. Woods. *Speech Understanding Systems: Final Report of a Study Group.* Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1971. Also Elsevier/North-Holland, Amsterdam, 1973.

Newell [73] Newell, A. Production systems: Models of control structures. In W. C. Chase (Ed.), *Visual Information Processing*, Academic Press, NY, 463-526.

Newell and Robertson [75] Newell, A. and G. Robertson. Some issues in programming multi-miniprocessors. *Behav. Res. Methods and Instr., 7,* 2, 75-86.

Reddy [66] Reddy, D. R. An approach to computer speech recognition by direct analysis of the speech wave. Doctoral Dissertation, AI Memo No. 43, Computer Science Dept., Stanford University, Stanford, CA, 1966.

Reddy and Vicens [68] Reddy, D. R. and Vicens, P. J. A procedure for segmentation of connected speech. *J. Audio Engr. Soc., 16,* 4 (1968).

Reddy, Erman, and Neely [70] Reddy, D. R., L. D. Erman, and R. B. Neely, The CMU speech recognition project. *Proc. IEEE System Sciences and Cybernetics Conf.*, Pittsburgh, PA, 1970.

Reddy [71] Reddy, D. R. Speech recognition: Prospects for the seventies. *Proc. IFIP 1971*, Ljubljana, Yugoslavia, Invited paper section I-5 to I-13.

Reddy, Erman, and Neely [72] Reddy, D. R., L. D. Erman, and R. B. Neely, A mechanistic model of speech perception. *Computer Science Research Review 1971-72*, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, 1972, 7-15.

Reddy, Erman, and Neely [73] Reddy, D. R., L. D. Erman, and R. B. Neely, A model and a system for machine recognition of speech. *IEEE Trans. Audio and Electroacoustics*, AU-21, 3, 1973, 229-238.

Reddy, Erman, Fennell, and Neely [73] Reddy, D. R., L. D. Erman, R. D. Fennell, and R. B. Neely, The Hearsay speech understanding system: An example of the recognition process. *Proc. 3rd Inter. Joint Conf. on Artificial Intel.*, Stanford, CA, 1973, 185-193.

Reddy [75] Reddy, D. R. (Ed.), *Speech Recognition: Invited Papers of the IEEE Symposium*. April 15-19, 1974, Pittsburgh, PA, Academic Press, NY, 1975.

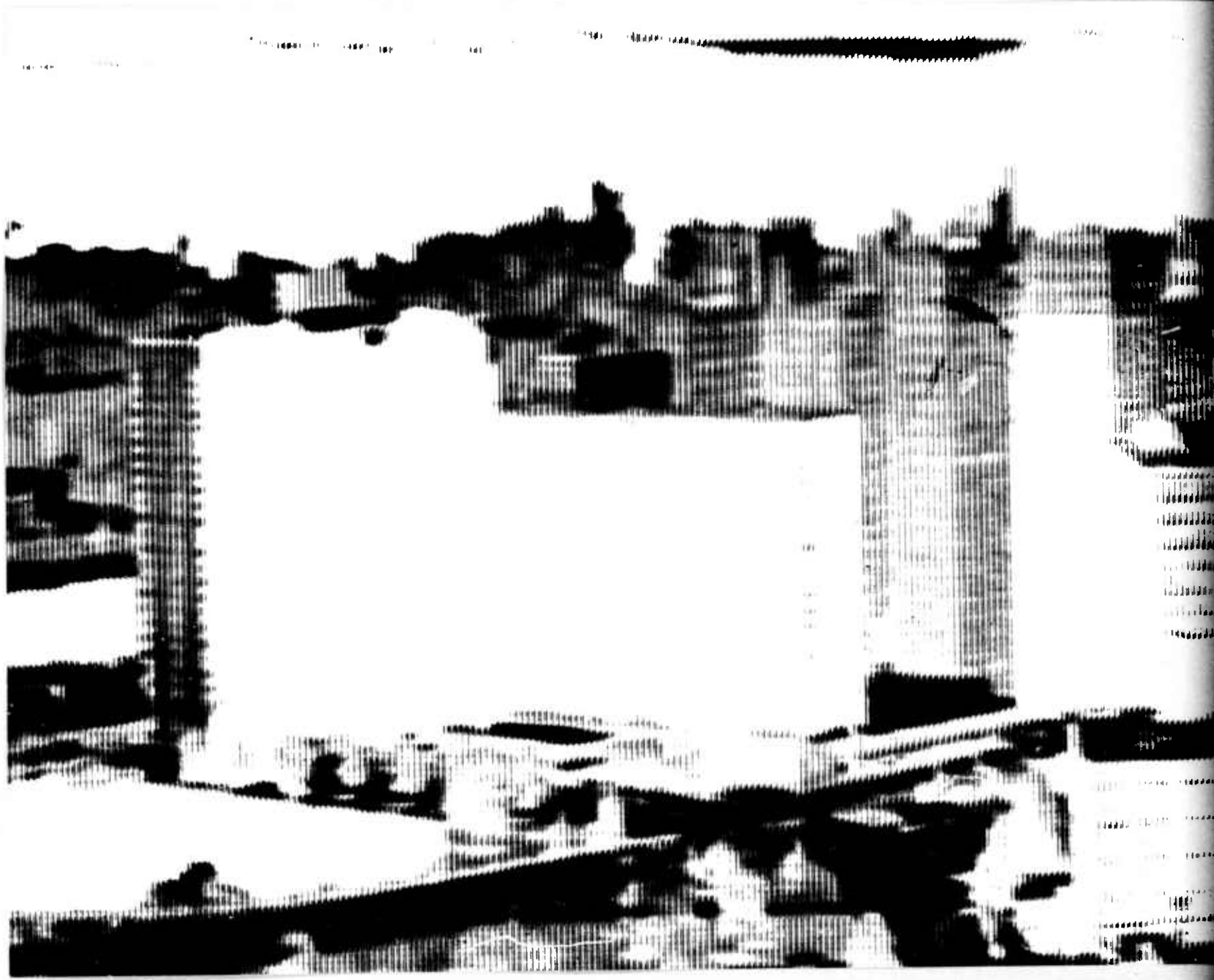
Shockey and Erman [74] Shockey, L. and L. D. Erman. Sub-lexical levels in the Hearsayll speech understanding system. In Erman [74b], 208-210.

Simon [66] Simon, H. A. Scientific discovery and the psychology of problem solving. *Mind and Cosmos: Essays in Contemporary Science and Philosophy*, Series in Philosophy of Science, University of Pittsburgh, Pittsburgh, PA, (1966), 3, 22-40.

Simon [71] Simon, H. A. The theory of problem solving. In *Information Processing 71*, North-Holland, 1971, 261-277.

VanLehn [73] VanLehn, K. A. SAIL User Manual. Memo AIM-204, Stanford Artificial Intelligence Laboratory, Stanford University, Stanford, CA, 1973.

Vicens [69] Vicens, P. Aspects of speech recognition. Doctoral Dissertation, Report CS-127, Computer Science Dept., Stanford University, Stanford, CA, 1969.



Strict Lower and Upper Bounds on Iterative Computational Complexity

Joseph F. Traub and Henryk Woźniakowski

1. Introduction

Complexity is a measure of cost. The relevant costs depend on the model under analysis. The costs may be taken as units of time (in parallel computation), number of comparisons (in sorting algorithms), size of storage (in large linear systems), or number of arithmetics (in matrix multiplication). Of course a number of different costs may be relevant to a model.

One can analyze the complexity of an algorithm, of a class of algorithms, or of a problem. The subject dealing with the complexity of an algorithm is usually called "Analysis of Algorithms." The subject dealing with the analysis of a class of algorithms or of a problem is called computational complexity.

Computational complexity comes in many flavors depending on the class of algorithms, the problem, and the costs. We limit ourselves here to mentioning three types of computational complexity. In each of these the costs are taken as the arithmetic operations. *Algebraic computational complexity* deals with a problem and a class of algorithms which solve the problems at finite cost. Typically the problem belongs to a class of problems which is indexed by an integer n. Let $comp(P_n)$ be the complexity of solving the nth problem in the class. We are interested in lower bounds $L(P_n)$ and upper bounds $U(P_n)$ on $comp(P_n)$,

$$(1.1)\quad L(P_n) \le comp(P_n) \le U(P_n).$$

The upper bounds are obtained by exhibiting an algorithm for solving P_n with complexity $U(P_n)$. Lower bounds are obtained by theoretical considerations and "non-trivial" lower bounds are difficult to obtain. For example if P_n is the problem of multiplying two n by n matrices and if the cost of each arithmetic operation is taken as unity then

$$0(n^2) \le comp(P_n) \le 0(n^{\beta}),\ \beta = \lg 7.$$

(We use lg to represent $\log_2$.) Borodin and Munro [75] survey the state of knowledge in algebraic complexity.

Exact solutions of "most" problems in science, engineering, and applied mathematics cannot be obtained with finite cost even if infinite-precision arithmetic is assumed. Indeed linear problems and evaluation of rational functions which can be solved at finite cost are the exception. Even when the problem can be solved rationally, we may choose to solve it by iteration. An example is the solution of large sparse linear systems. Typically, non-linear problems cannot be solved at finite cost.

We call the branch of complexity theory that deals with non-finite cost problems *analytic computational complexity*. Often the algorithms are iterative and we then refer to *iterative computational complexity*. See Traub [75] for papers presented at a CMU Conference on Analytic Computational Complexity.

In this paper we propose a new methodology for iterative computational complexity. Our aim is to create at least a partial synthesis between iterative complexity and other types of complexity.

A basic quantity in iterative complexity has been the *efficiency index* of an algorithm or class of algorithms. In this paper we introduce a new quantity, the *complexity index*, which is the reciprocal of the efficiency index. The complexity index is directly proportional to the complexity of an algorithm or class of algorithms. We show under what conditions the complexity index is a good measure of complexity. Our methodology is *non-asymptotic* in the number of iterations. Earlier analyses of complexity applied only as the number of iterations went to infinity and this is not of course realistic in practice.

We summarize the contents of this paper. In Section 2 we analyze a simplified model of the errors of an iterative process and show that complexity is the product of two factors, the complexity index and the error coefficient function. Bounds on the error coefficient function are derived in the following Section and used to derive rigorous conditions for comparing the complexity of two different algorithms. In Section 4 we show how the results of the simple model can be applied to a realistic model of one-point iteration. Lower and upper bounds on the complexity index for several important classes of iterations appear in Section 5. In a short concluding Sec-

tion we state the extensions and generalizations to be reported in future papers.

2. Basic Concepts

We analyze algorithms for the following problem. Let f be a non-linear real or complex scalar function with a simple zero α. Let x_0 be given and let an algorithm φ generate a sequence of approximations $x_1,...,x_k$ to α. We terminate the algorithm when x_k is a sufficiently good approximation to α. This will be made precise below.

The appropriate setting for this investigation is to consider f as a non-linear operator on a Banach space of finite or infinite dimension. Since many of the basic ideas can be illustrated when f is a non-linear scalar function we shall assume throughout this paper that this holds. We must remark however
32 that some of the most interesting and important results deal with the dependence of complexity on problem dimension and we do not deal with that dependence here.

Let $e_i > 0$ represent some measure of the error of x_i. For example, e_i might represent

$$|x_i-\alpha|, \text{ the absolute error}$$

$$\left|\frac{x_i-\alpha}{|\alpha|}\right|, \text{ the relative error}$$

$$|f(x_i)|, \text{ the residual.}$$

Assume that the e_i satisfy the error equation

$$(2.1)\quad e_i = A_i e_{i-1}^p,\ p \geq 1,\ i = 1,2,...,k.$$

We call p the *non-asymptotic order* and A_i the *error coefficient*. We require $0 < L \leq A_i \leq U < \infty$ for all values of e_0 including the possibility that e_0 be arbitrarily small. Then p is unique. Many iterations satisfy the model given by (2.1). In Section 6 we mention extensions to this model.

EXAMPLE 2.1. Let the algorithm be Newton-Raphson iteration and let e_i denote the absolute error. Then

$$p = 2,\ A_i = \left|\frac{f''(\eta_i)}{2f'(x_i)}\right|,$$

where η_i is in the interval spanned by α and x_i.

We simplify the model of (2.1) and show what kind of results may then be obtained. In Section 4 we return to the analysis of (2.1). Let

$$(2.2)\quad e_i = Ae_{i-1}^p,\ p \geq 1,\ i = 1,...,k.$$

We call this the *constant error coefficient model* while (2.1) is the *variable error coefficient model*.

We consider first the case $p > 1$. It is easy to verify that

$$(2.3)\quad e_i = e_0\left(\frac{1}{w_p}\right)^{p^i-1},\ i = 0,...,k,$$

where

$$(2.4)\quad w_p = \frac{1}{A^{\frac{1}{p-1}} e_0}.$$

Choose ϵ', $0 < \epsilon' < 1$, and let k be the smallest index for which $e_k \leq \epsilon' e_0$. Define $\epsilon \leq \epsilon'$ so that

$$(2.5)\quad e_k = \epsilon e_0.$$

ϵ' is a basic parameter which measures the increase in precision to be obtained in the iteration. We choose ϵ to avoid ceiling and floor functions later in this paper. It is convenient to assume $\epsilon \leq 2^{-2}$ (we use this in Theorem 3.1) but this is non-restrictive in practice.

From (2.3), (2.5),

$$(2.6)\quad \epsilon = \left(\frac{1}{w_p}\right)^{p^k-1},$$

and it follows that

$$(2.7)\quad k = \frac{g(w_p)}{\lg p},$$

where

$$(2.8)\quad g(w_p) = \lg\left(1 + \frac{t}{\lg w_p}\right),\ t = \lg(1/\epsilon).$$

This is independent of the logarithm base but it is convenient to take all logarithms to base 2. Then if e_i is the relative error, t measures the number of bits to be gained in the iteration.

We denote the complexity of iteration i by c_i. In this paper we assume $c_i = c$ is independent of i. We defer a discussion of the estimation of c until Section 5. The important case of variable cost will be considered in a future paper. We define the complexity of the algorithm by

$$(2.9)\quad \text{comp} = ck.$$

Then from (2.7), (2.8),

$$(2.10)\quad \text{comp} = zg(w_p)$$

where we define

$$(2.11)\quad z = \frac{c}{\lg p}.$$

as the *complexity index*.

We call g the *error coefficient function*. Equation (2.10) will be fundamental in our further analysis.

We have decomposed complexity into the product of two factors. The complexity index, which is independent of both the error coefficient and the starting error, is relatively easy to compute for any given algorithm. (However, lower bounds on the complexity index for classes of algorithms require upper bounds on order which is a difficult problem only solved for special cases (Kung and Traub [73], Meersman [75] and Woźniakowski [75b]).) We shall show, in a sense to be made precise in the next section, that the error coefficient is insensitive for a large portion of its domain and that complexity is determined primarily by the complexity index. We shall also show there are cases where complexity is determined primarily by the error coefficient function.

The complexity index is the reciprocal of a quantity called the efficiency index which has played an important role in iterative complexity. See, for example, Traub [64, Appendix C], Traub [72], Paterson [72] and Kung [73a]. Since complexity varies directly with the complexity index we feel that the complexity index rather than the efficiency index, should be basic.

We have been considering the case $p > 1$. For completeness we write down the case $p = 1$. Then $e_i = Ae_{i-1}$, $i = 1, 2, \ldots, k$ and $e_k = A^k e_0 = \epsilon e_0$. Hence

$$(2.12)\quad k = \frac{t}{\lg(1/A)},\ \text{comp} = \frac{ct}{\lg(1/A)}.$$

We shall not pursue the case $p = 1$ further and shall assume for the remainder of this paper that $p > 1$, unless we state otherwise.

3. Bounds on the Error Coefficient Function

We turn to an analysis of the error coefficient function which is one of the two factors which determines the complexity in (2.10). To see which values of w_p are of interest, note that from (2.3), $e_k < e_0$ iff $w_p > 1$. From the definition of k it is clear that $k \geq 1$ and hence from (2.7), (2.8),

$w_p \leq (1/\epsilon)^{1/(p-1)}$. Hence we assume

$$(3.1)\quad 1 < w_p \leq (1/\epsilon)^{1/(p-1)} = 2^{t/(p-1)}.$$

Generally w_p depends on p. For many classes of iterations

$$(3.2)\quad a^{p-1} \leq A \leq b^{p-1}.$$

Then

$$1/(ae_0) \geq w_p \geq 1/(be_0)$$

and the bounds on w_p are independent of p. If (3.2) holds for a class of iterations ϕ we shall say the class is *normal*. An example of a normal class of iterations may be found in Woźniakowski [75b]. To simplify notation we shall henceforth write w_p as w whether or not we are dealing with a normal class.

Now, g(w) is a monotonically decreasing function and

$$\lim_{w \to 1^+} g(w) = \infty, \quad \lim_{w \to \infty} g(w) = 0.$$

To study the size of g(w) we somewhat arbitrarily divide the range of w, given by (3.1), into three subranges. The bounds are not the sharpest we can obtain.

$1 < w \leq 2$. Since $g(w) = \lg(t + \lg w) - \lg\lg w$ and $0 < \lg w \leq 1$, we conclude

$$\lg t - \lg\lg w < g(w) \leq \lg(1+t) - \lg\lg w.$$

$2 \leq w \leq t$. Since $g(w)$, $g(2) = \lg(1+t)$, $g(t) > \lg t - \lg\lg t$, we conclude

$$\lg t - \lg\lg t < g(w) \leq \lg(1+t).$$

$t \leq w \leq 2^{t/(p-1)}$, $2^{t/(p-1)} \geq t$. Then

$$\lg p \leq g(w) < 1 + \lg t - \lg\lg t.$$

To get some feel for the length of these subranges, observe that if e_i represents relative error then in single-precision computation on a "typical" digital computer we might take $\epsilon = 2^{-32}$. Then $t = 32$ and if $p = 2$, then $2^{t/(p-1)} = 2^{32}$.

From the bounds on the error coefficient function and (2.10) we immediately obtain the following bounds on complexity.

THEOREM 3.1. If $1 < w \leq 2$,

$$(3.3)\quad z(\lg t - \lg\lg w) < \text{comp} \leq z(\lg(1+t) - \lg\lg w).$$

If $2 \leq w \leq t$,

$$(3.4)\quad z(\lg t - \lg\lg t) \leq \text{comp} \leq z\lg(1+t).$$

If $t \leq w \leq 2^{t/(p-1)}$, (with $2^{t/(p-1)} \geq t$),

$$(3.5)\quad c \leq \text{comp} < z(1 + \lg t - \lg\lg t).$$

We discuss some of the implications of this Theorem. As w approaches unity, then for ϵ fixed, comp $\sim -z\lg\lg w$. In this case the effect of the error coefficient A and the initial error e_0 cannot be neglected.

Complexity depends more on the nearness of w to unity than of ϵ to zero. To see this, observe that if $2 \le w \le t$, comp $\sim$ z lg lg $(1/\epsilon)$ = $comp_1$ while if $1 < w \le 2$, comp $\sim$ z(lg lg $(1/\epsilon)$ - lg lg w) = $comp_2$. Let $\epsilon = 2^{-2^j}$, $w-1=2^{-2^j}\ln 2$. Then $comp_1 = jz$, $comp_2 \sim z(j+2^j)$.

Note that for any $p > 1$ the complexity of an iteration can be greater than if $p = 1$ (see (2.12)) provided w is sufficiently close to unity.

For any $w \ge 2$, complexity is bounded from above by z lg(1+t) and is therefore independent of the error coefficient A and the initial error e_0. For $w \ge 2$, complexity is insensitive to w and we need only crude bounds on w.

For $2 \le w \le t$,

$$1-\lg\lg t/\lg t \le \text{comp}/(z\lg t) \le 1+\lg(1+t^{-1})/\lg t$$

Therefore

$$1+o(1) \le \text{comp}/(z\lg t) \le 1+o(1)$$

and we conclude that on the interval [2,t] we have, for t large, very tight bounds on comp with

(3.6) comp $\sim$ z lg lg $1/\epsilon$.

This should be compared with the case $p = 1$ (see (2.12)) where comp varies as lg $1/\epsilon$.

We have taken $w = 2$ as one of our endpoints for convenience but this is of course arbitrary. Any value of w sufficiently far from unity will do. If $w = 2^{1/\nu}$ then $g(w) = \lg(1+\nu t)$. Then the effect of the nearness of w to unity and of ϵ to zero are equal if $\nu = t$, that is if $w = 2^{1/t}$. For this choice of w, comp = $z\lg(1+t^2)$ $\sim 2z\lg t = 2z\lg\lg 1/\epsilon$.

We have chosen the sub-ranges of w so that the endpoints are simple. We could also choose values of w that make the complexity formula simple. If

$w = 2^{t/(t^u-1)}$, $u \ge 1$, then comp = $uz\lg\lg(1/\epsilon)$, while if

$w = 2^{t/(t^{1/\nu}-1)}$, $\nu \ge 1$, then comp = $(1/\nu)z\lg\lg(1/\epsilon)$

We now consider the methodology for *comparing two iterations* which are governed by the constant error coefficient model (2.2) and decrease the final error by the same ϵ. Let w_i, z_i, $comp_i$, $i = 1,2$ denote the parameters of the two iterations. Then

$$\frac{comp_1}{comp_2} = \left(\frac{z_1}{z_2}\right)\frac{g(w_1)}{g(w_2)}.$$

Clearly if $z_1 \le z_2$ and $w_1 \ge w_2$ then $comp_1 \le comp_2$. We obtain bounds on $comp_1/comp_2$ for sub-ranges of the w_i. Using the bounds on complexity from the previous theorem we obtain

THEOREM 3.2. If $1 < w_1, w_2 \le 2$, then

(3.7)

$$\left(\frac{z_1}{z_2}\right)\left(\frac{\lg t-\lg\lg w_1}{\lg(1+t)-\lg\lg w_2}\right) < \frac{comp_1}{comp_2} < \left(\frac{z_1}{z_2}\right)\left(\frac{\lg(1+t)-\lg\lg w_1}{\lg t-\lg\lg w_2}\right)$$

If $1 < w_2 \le 2 \le w_1 \le t$, then

(3.8)

$$\left(\frac{z_1}{z_2}\right)\left(\frac{\lg t-\lg\lg t}{\lg(1+t)-\lg\lg w_2}\right) \le \frac{comp_1}{comp_2} < \left(\frac{z_1}{z_2}\right)\left(\frac{\lg(1+t)}{\lg t-\lg\lg w_2}\right).$$

If $2 \le w_1, w_2 \le t$, then

(3.9) $$\left(\frac{z_1}{z_2}\right)\left(\frac{\lg t-\lg\lg t}{\lg(1+t)}\right) < \frac{comp_1}{comp_2} < \left(\frac{z_1}{z_2}\right)\left(\frac{\lg(1+t)}{\lg t-\lg\lg t}\right).$$

We discuss some of the implications of this theorem. As $t \to \infty$, $comp_1/comp_2 \to z_1/z_2$ for any fixed values of w_1, w_2. The ratio z_1/z_2 has been the way that iterations have been compared (see Traub [64, Appendix C] where efficiency indices are used). Theorem 3.2 shows that z_1/z_2 can be a very poor measure of $comp_1/comp_2$; see for example (3.7).

Finally we observe that inequalities (3.7)-(3.9) can be rewritten to show when $comp_1 < comp_2$ or $comp_2 < comp_1$. For example, if $2 \le w_1, w_2 \le t$,

(3.10) $z_1 \le z_2\left(\frac{\lg t-\lg\lg t}{\lg(1+t)}\right)$, then $comp_1 < comp_2$.

4. The Variable Error Coefficient Model

We turn to the variable error coefficient model,

(4.1) $e_{i+1} = A_i e_i^p$.

A complete analysis of this model is beyond the scope of this paper. Here we confine ourselves to the very simple assumption

(4.2) $A_L \le A_i \le A_U$, $i = 1,\ldots,k$.

Let

$$w_L = \frac{1}{\frac{1}{A_L^{p-1}}e_0}, \quad w_U = \frac{1}{\frac{1}{A_U^{p-1}}e_0}.$$

Then

(4.3) $zg(w_L) \le comp \le zg(w_U)$.

Note that $w_U \le w_L$ and therefore (4.3) is compatible with g being a monotonically decreasing function. We can now draw conclusions from the constant coefficient model with A replaced by A_L or A_U.

EXAMPLE 4.1. Let α be a real zero and let J denote an interval centered at α. Assume f' does not vanish in J and let $x_0 \in J$ and such that

$$e_0 = |x_0 - \alpha| \le \frac{\min_{x \in J} |f'(x)|}{\max_{x \in J} |f''(x)|} = \frac{1}{2A_U}.$$

Then by Example 2.1, for Newton-Raphson iteration, $w_U \ge 2$ and *a priori*

(4.4) $comp \le c \lg(1+t)$.

The value of c is discussed in Section 5. Note that a sufficient condition for convergence is

$$e_0 < 1/A_U$$

but with only this condition, complexity could be extremely large.

EXAMPLE 4.2. We seek to calculate $a^{1/2}$, that is solve $f(x) = x^2 - a$. Let $a = 2^m \lambda^2$, m even, $1/2 \le \lambda^2 < 2$. Then $a^{1/2} = 2^{m/2}\lambda$, $(1/2)^{1/2} \le \lambda < 2^{1/2}$. We use Newton-Raphson iteration,

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{\lambda^2}{x_i}\right)$$

Then $A_i = 1/(2x_{i-1})$. If $x_0 > \lambda$, then

$$A_L = 1/(2x_0) \le A_i < 1/(2\lambda) = A_U, \quad i = 1,\ldots,k.$$

Hence

$$w_L = \frac{2}{1-\lambda/x_0}, \quad w_U = \frac{2\lambda/x_0}{1-\lambda/x_0}.$$

Let $x_0 = 2^{1/2}$. Then $w_U \ge 2$ and $comp \le c \lg(1+t)$. To derive a lower bound on complexity one must make an assumption about the closest machine-representable number to $2^{1/2}$. We do not pursue that here.

5. Bounds on the Complexity Index

We have shown that provided w is not too close to unity, then for fixed ϵ, complexity depends only on the complexity index z. In this section we turn our attention to the complexity index.

Recall that $z = c/\lg p$. We begin our analysis of z by considering the cost per step, c. We distinguish between two kinds of problems.

We say a problem is *explicit* if the formula for f is given explicitly. For example, the calculation of $a^{1/2}$ by solving $f = x^2 - a$ is an explicit problem. The complexity of explicit problems has been studied by Paterson [72] and Kung [73a], [73b]. (Paterson and Kung take the efficiency index as basic.) We do not treat explicit problems here.

We say a problem is *implicit* if all we know about f are certain functionals of f. Classically the functionals are f and its derivatives evaluated at certain points. These functionals may be thought of as black boxes which deliver an output for any input. Kacewicz [75] has shown that integral functionals are of interest. The question of what functionals may be used in the solution of a problem are beyond the scope of this paper. We confine ourselves to implicit problems for the remainder of this paper.

We assume the same set of functionals is used at each step of the iteration. The set of functionals used by an iteration algorithm ϕ is called the information set *N*. Woźniakowski [75a] gives many examples of *N*. Let the *information complexity* $u = u(f,N)$ be the cost of evaluating functionals in the information set *N* and let the *combinatory complexity* $d = d(\phi)$ be the cost of combining functionals (see Kung and Traub [74b]). We assume that each arithmetic operation costs unity and denote the number of operations for one evaluation of $f^{(j)}$ by $c(f^{(j)})$. The following simple example may serve to illustrate the definition.

EXAMPLE 5.1. Let ϕ be Newton-Raphson iteration $x_{i+1} = \phi(x_i) = x_i - f(x_i)/f'(x_i)$, $i = 0,\ldots,k-1$. Then $N = \{f(x_i), f'(x_i)\}$, $u(f,N) = c(f) + c(f')$, $d(\phi) = 2$.

Up to this point we have illustrated the concepts with algorithms. Computational complexity deals with classes of algorithms and we turn to our central concern, lower and upper bounds on classes of algorithms. As usual the difficult problem is obtaining lower bounds. Good lower bounds may be obtained from good lower bounds on cost and good upper bounds on order. The problem of maximal order is a difficult one about which a great deal has been recently learned (Meersman [75], Woźniakowski [75a], [75b]). Part of the mathematical difficulty of the subject deals with the problem of maximal order. Note however that maximal order

does not necessarily minimize complexity; we deal with this in a future paper. Upper bounds are obtained from algorithms. An interesting question here is a good upper bound on the combinatory complexity of a class of algorithms. Brent and Kung [75] have obtained a surprising new upper bound, $0(r \lg n)$, on the combinatory complexity on a family of nth order one-point iterations based on inverse interpolation.

It is convenient to index our algorithms by n, the number of elements in the information set N. We illustrate the issues with two examples.

EXAMPLE 5.2. Let ϕ_n denote any one-point iteration with $N = \{f(x_i), f'(x_i), \ldots, f^{(n-1)}(x_i)\}$. Let $c_f = \min_i c(f^{(i)})$. Then $u(f,N) \geq nc_f$. For simplicity we use the linear lower bound $d(\phi_n) \geq n-1$. (No non-linear lower bound is known.) A sharp upper bound on the order of one-point iteration (Traub [64], Kung and Traub [74a]) is $\rho \leq n$. Hence

$$z(\phi_n, f) \geq \frac{nc_f + n - 1}{\lg n},$$

$$z(\phi_n, f) \geq \frac{nc_f + n - 1}{\lg n} \geq \frac{3c_f + 2}{\lg 3},$$

provided only that $c_f \geq 4$ (Kung and Traub [74b]). Hence for any one-point iteration with $w_L \leq t$

$$\text{(5.1)} \quad \text{comp} \geq \frac{3c_f + 2}{\lg 3}(\lg t - \lg\lg t).$$

On the other hand there exists a one-point iteration which uses f, f', f'' and such that $p = 3$. Hence if $w_U \geq 2$,

$$\text{(5.2)} \quad \text{comp} \leq \frac{c(f) + c(f') + c(f'')}{\lg 3} \lg(1+t).$$

For problems such that $c(f) \simeq c(f') \simeq c(f'') \simeq c_f$ the lower and upper bounds of (5.1) and (5.2) are close together.

EXAMPLE 5.3. Kung and Traub [74a] show there exists an iteration ψ_n for which the information set N consists of n evaluation of f with $p(\psi_n) = 2^{n-1}$ and $d(\psi n) = (3/2)n^2 + (3/2)n - 7$. Hence

$$z(\psi_n) = \frac{nc(f) + \frac{3}{2}n^2 + \frac{3}{2}n - 7}{n-1}$$

The complexity index is minimized (Kung and Traub [74b]) at $n^* = \text{round}[1 + (\frac{2}{3}(c(f)-4))^{1/2}] = 0(c(f))^{1/2}$ and

$$z(\psi_{n^*}) = c(f)/\left(1 + \frac{\xi}{(c(f))^{1/2}}\right), \quad \xi > 0.$$

It would only be reasonable to use this high an order iteration for very small ϵ. Assume $t >> p^* = 2^{n-1}$.

Observe that $z(\psi_n)$ is a very "flat" function of n. Thus $z(\psi_3) = (3/2)c(f) + 11/2$ and comparing this with $z(\psi_{n^*})$ shows we can gain only another $(1/2)c(f)$.

Let Φ denote the class of all multipoint iterations for which $w_U \geq 2$. Then

$$\text{comp}(\Phi) \leq c(f)\lg(1+t)/\left(1 + \frac{\xi}{(c(f))^{1/2}}\right).$$

We can obtain a lower bound on the complexity of the class of multipoint iterations by using an upper bound on the maximal order of any multipoint iteration and a lower bound on the combinatorial complexity. Kung and Traub [74a] conjecture that *any* iteration without memory which uses n pieces of information per step has order $p \leq 2^{n-1}$. This conjecture seems difficult to prove in general (Woźniakowski [75b]) but has been established for many important cases (Kung and Traub [73], Meersman [75], and Woźniakowski [75b]).

6. Summary and Extensions to the Model

We have constructed a non-asymptotic theory of iterative computational complexity with strict lower and upper bounds. In order to make the complexity ideas as accessible as possible we have limited ourselves to scalar non-linear problems. The natural setting for this work is in a Banach space of finite or infinite dimension and we shall do our analysis in this setting in a future paper. We have focused on the simplified model $e_i = Ae_{i-1}^p$. More realistic models include some of the following features;

1. $e_i = A_i e_{i-1}^p$ under various assumptions on the structure of A_i.
2. $e_i = A_i e_{i-1}^{p_1} \ldots e_{i-m}^{p_m}$. This is the appropriate model for iterations with memory.
3. Variable cost per iteration, c_i.
4. Include round-off error. Then e_i will not converge to zero.

We plan to analyze these more realistic models in the future. We also intend to investigate additional basic properties of complexity. Our various results will be used to analyze the complexity of important problems in science and engineering.

Acknowldgement

We thank H. T. Kung for his comments on this paper.

References

Borodin and Munro [75] Borodin, A. and Munro, I. *The Computational Complexity of Algebraic and Numeric Problems*. American Elsevier, New York, N.Y., 1975.

Brent and Kung [75] Brent, R. P. and Kung, H. T. In preparation.

Kacewicz [75] Kacewicz, B. The use of integrals in the solution of non-linear equations in N dimensions. To appear in *Analytic Computational Complexity*, Traub, J. F. (ed.). Academic Press, New York, N.Y., 1975.

Kung [73a] Kung, H. T. A bound on the multiplicative efficiency of iteration. *Journal of Computer and System Sciences 7* (1973), 334-342.

Kung [73b] Kung, H. T. The computational complexity of algebraic numbers. *SIAM J. Numer. Anal. 12* (1975), 89-96.

Kung and Traub [73] Kung, H. T. and Traub, J. F. Optimal order and efficiency for iterations with two evaluations. Technical report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa. 1973. To appear in *SIAM J. Numer. Anal.*

Kung and Traub [74a] Kung, H. T. and Traub, J. F. Optimal order of one-point and multipoint iteration. *Journal of the Association for Computing Machinery 21* (1974) 643-651.

Kung and Traub [74b] Kung, H. T. and Traub, J. F. Computational complexity of one-point and multipoint iteration. *Complexity of Computation*, Karp, R. (ed.). American Mathematical Society, Providence, R.I., 1974, 149-160.

Meersman [75] Meersman, R. Optimal use of information in certain iterative processes. To appear in *Analytic Computational Complexity*, Traub, J. F. (ed.). Academic Press, New York, N.Y., 1975.

Paterson [72] Paterson, M. S. Efficient iterations for algebraic numbers. *Complexity of Computer Computations*, Miller, R. and Thatcher, J. W. (eds.). Plenum Press, New York, N.Y., 1972, 41-52.

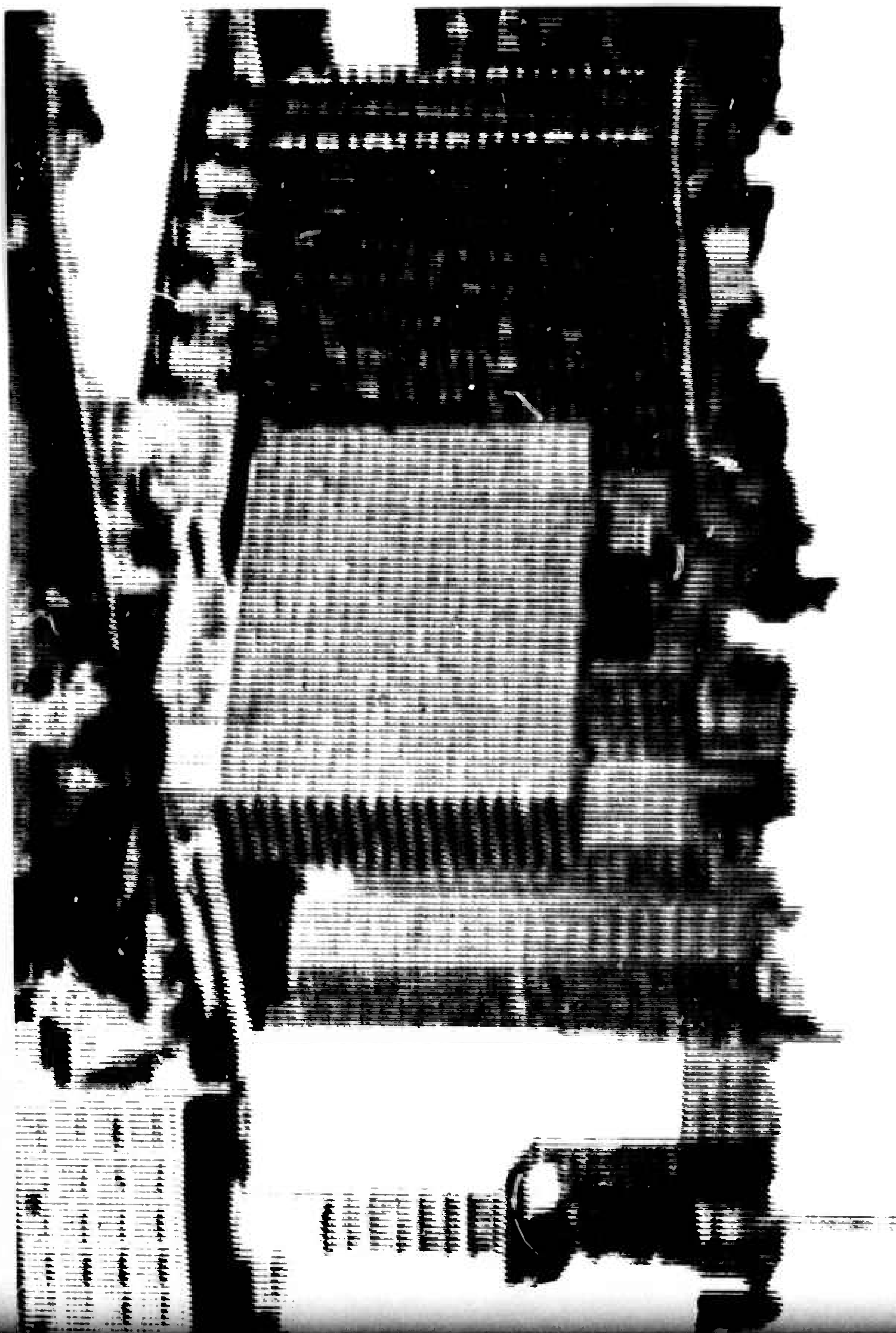
Traub [64] Traub, J. F. *Iterative Methods for the Solution of Equations*. Prentice-Hall, Englewood Cliffs, N.J., 1964.

Traub [72] Traub, J. F. Computational complexity of iterative processes. *SIAM Journal of Computing 1* (1972), 167-179.

Traub [75] Traub, J. F. (ed.). *Analytic Computational Complexity*. Academic Press, New York, N.Y., 1975.

Woźniakowski [75a] Woźniakowski, H. Generalized information and maximal order of iteration for operator equations. *SIAM J. Numer. Anal. 12* (1975), 121-135.

Woźniakowski [75b] Woźniakowski, H. Maximal order of multipoint iterations using n evaluations. To appear in *Analytic Computational Complexity*, Traub, J. F. (ed.). Academic Press, New York, N.Y., 1975.



The CMU RT-CAD System: An Innovative Approach to Computer Aided Design

Mario R. Barbacci and Daniel P. Siewiorek

I. Introduction and Motivation

As technology has evolved the primitive components available to a digital system designer have changed dramatically. Twenty-five years ago the designer constructed his systems out of circuit level components such as resistors and diodes. Subsequently switching circuit level components, as represented by gates and flip-flops, became available as small scale integration (SSI) components. With the introduction of medium scale integration (MSI) register transfer level components appeared: arithmetic and logic units registers, shift registers, etc. The advent of large scale integration (LSI) has made memories and even processors primitive components from which systems are designed. Two trends can be observed from this technological evolution: 1.) primitive components continue to increase in complexity and 2.) the rate of introduction of new components continues to increase.

In response to the first trend, designers have been limiting their excursions into switching circuit level design to only small portions of the system (e.g., bus controllers, etc.). In some register transfer level module sets (Bell [72], Clark [67]) these excursions have been completely eliminated.

Because of the second trend, rapid technology evolution, there is a need to shorten the delay time between the introduction of a technology and its effective use in new computing systems. Also, as technology changes so does its cost. The design process must, therefore, be accelerated if the potentiality of the improving technology is to be realized.

This paper describes a set of design programs developed at Carnegie-Mellon. The ultimate goal is to minimize the effect of changing technology by building a Computer Aided Design System that implements a technology-relative design process.

II. Overview of the Automatic Design Process

Given the complexity of a digital system, designers have sought to develop automatic means to reduce the cost and time of the design process. The objective was to relieve engineers of repetitive, time consuming tasks such as:

(1) The generation of detailed design information (gate and chip types, etc.)
(2) The control of changes in the design documents
(3) The checking of the system for electrical, logical, and physical compatibility (fan-out limits, etc.)
(4) The generation of detailed manufacturing information (chip placement, board layout, etc.)

This early view of design automation limited itself to filling the gap between the low-level design specifications and the manufacturing data. Behavioral specifications were in the form of Boolean equations and the design programs translated them into their equivalent logic diagrams and wiring lists. Most of the synthesis algorithms at this level dealt with the problem of reducing or simplifying the Boolean equations (Breuer [72]).

Subsequent efforts were directed towards a system capable of accepting a higher level of behavioral description, although still oriented towards a gate level implementation (Darringer [69], Friedman [69]).

Current design automation effort is shifting from implementation in terms of the switching circuit level to implementation in terms of the Register Transfer level. Register Transfer level simulators have preceded this trend by several years (Darringer [69], Mesztenyi [68], Parnas [67], Rozenberg [71]). The closeness of RT level descriptions to conventional programming accounts for this early success. Register Transfer level descriptions are easy to transliterate into executable programs in a conventional programming language (e.g., FORTRAN, Algol, etc.), thus providing inexpensive and fast simulation (although in many cases RT languages are compiled directly). Register Transfer level synthesis algorithms have been less successful. A few programs have been developed that take an RT level description as input and compile it directly into a known set of RT level hardware modules (Chartran, AHPL). Figure 1 depicts a typical RT design automation system. The RT level description serves as input to several software modules. Syntax checking insures a well formed description. Static checking attempts to locate logical design

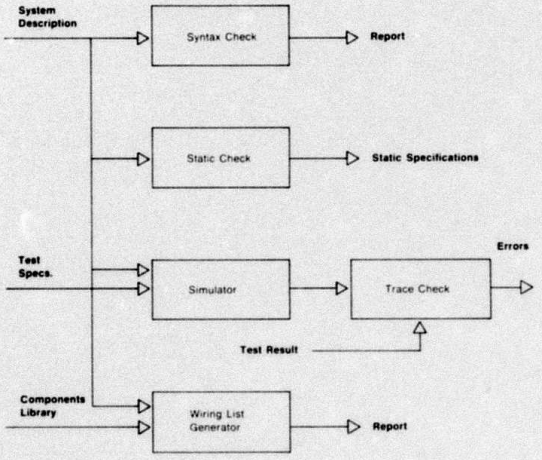


Figure 1
A Conventional Register Transfer Level Design Automation System

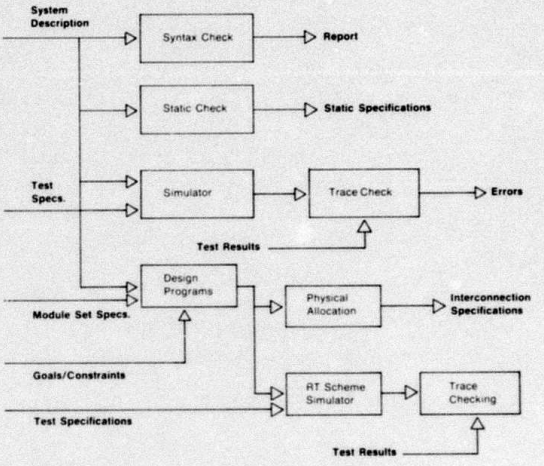


Figure 2
An Augmented Register Transfer Level Design Automation System

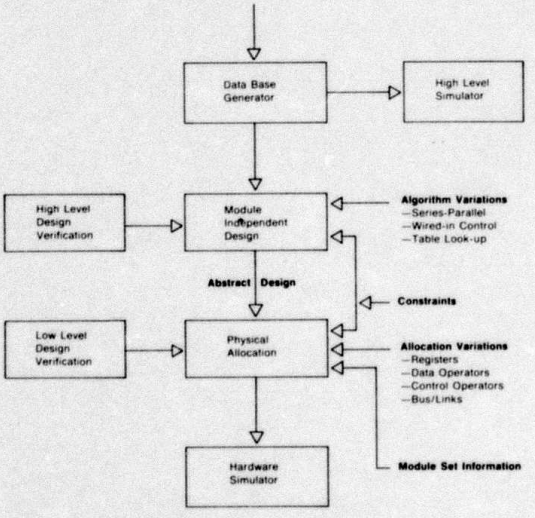


Figure 3
The Design Process in a RT Level Design System

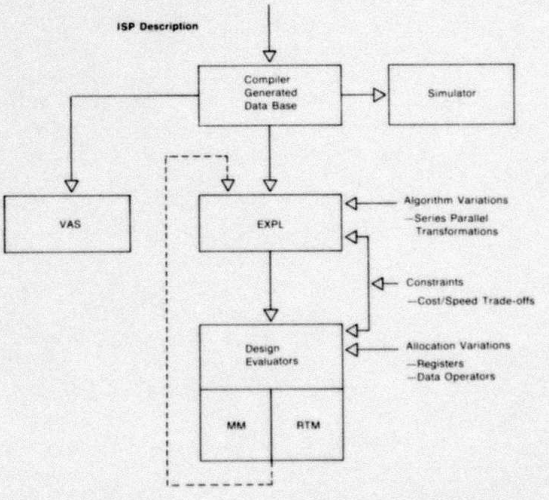


Figure 4
The Design Process in the CMU RT-CAD System (June 1975)

errors (such as deadlocks, redundancy, etc.). The simulator is used to debug the design dynamically. Finally, the description is cast into hardware via the wiring list generator.

The essential feature lacking in conventional RT Design Automation (DA) systems, and DA systems in general, is the exploitation of alternative implementations derived from the initial behavioral specification. Consider the augmented DA system depicted in Figure 2. The inputs are the RT level description and designer given constraints. The output is the specification/simulation of the hardware that attempts to optimize the system according to the design constraints. By allowing the description of various module sets the system can perform design relative to technology thus speeding up the incorporation of new technology into the design process. Also, such a system will allow experimentation with multiple module sets, each tailored to a specific class of problems. The system would also facilitate the design of the module sets themselves. Since the system operates on a symbolic description of the modules, a non-existing module set can be fed to the system for experimentation purposes. Such experiments will point out the advantages and disadvantages of the proposed module set.

At this point it would be instructive to describe the order in which the DA programs are typically used in the design process. This will serve to place subsequent discussions in perspective. Given a computational task, there are usually several algorithms that can be employed. The algorithm that is selected by the designer is described to the design automation system (Figure 3) and placed in a data base. Subsequently all design automation programs will use this data base. A high level simulator can execute from the data base to facilitate user debugging of the initial description.

Next some evaluation and reshaping of the algorithm is undertaken. Analysis tools have been developed to check the algorithm for well-formness (e.g., deadlock conditions, etc.) (Huen [75]). Perturbations of the basic algorithm can also be attempted such as: series-to-parallel transformations, replacing loop counters by wired-in control, and using table look-up in lieu of computing the value of functions. Thus attempts are made to first bind those design decisions with global implications. While these perturbations can be performed independent of the physical design, the evaluation of their ultimate desirability may depend upon the module set used to implement the final, physical design.

Finally, the actual physical design is performed in terms of RT level modules. The module set can be selected from a library of module sets or a user described set. At this level several forms of allocation variations are encountered:

- Registers. Determine the allocation of the abstract variables to registers and memory.
- Data operators. Determine the number of operators of each type in the design.
- Control. Select control schema from among unary state encoding, binary state encoding, microprogram control, etc.
- Bus-Link clustering. Many RT designs start with a set of registers for variables and interconnect them with links to operators (add, shift, multiply, etc.). After a point the interconnections between certain registers and operators become numerous enough to warrant replacement by a bus.
- Operator interconnection. The interconnection of operators has been shown to have a significant effect on the test generation effort required for the physical implementation (Stephenson [74]).

The signal level design verifier can be used to analyze the intermodule signal relationships in proposed module sets. Even well-established module sets have exhibited deadlock behavior in what appear to be straightforward interconnections (Huen [75]).

A first version of the above system has been implemented at Carnegie-Mellon and is shown in Figure 4. The behavioral specifications of the system to be designed are provided in terms of the ISP language (Bell [71], Barbacci [75]). The compiler produces an "object"[1] program which is then loaded into the data base and manipulated by different design programs.

The next five sections will treat the applications programs in detail. Section III described EXPL, a module independent design program that examines series-parallel variations in the original algorithm. The following section presents the physical allocators for two existing RT module sets—RTMs and Macromodules. Section V discusses the heuristics used by EXPL to explore the design space. Sample

---

1 The compiler produces an "object" program in terms of a set of Register Transfer level primitive operations. This program appears in the form of an executable BLISS (Wulf [71]) program where each Register Transfer operation is represented by a call to a user-provided subroutine. By changing the set of subroutines, the compiler can support many diverse activities. The creation of the data base is, in fact, done by a specific set of subroutines. The compiler and the language are therefore independent of the applications. The uniform compiler output and the flexibility of the subroutine-call mechanism has simplified the interfacing to other application programs.

design spaces, examples of the application of the heuristics, and some observations are presented in section VI. Section VII concludes the discussion of the existing system by briefly outlining the remaining applications programs.

III. Automatic Design Space Exploration

EXPL (Barbacci [73]) takes as input the object code produced by the ISP compiler, together with a set of user given speed/cost constraints/tradeoffs. The compiler output is used to generate a graph representation of the behavior of the system. Subsequently, various series-to-parallel and parallel-to-series transforms on the graph are attempted to establish a new design. Several alternative designs are generated and passed to module set evaluators which complete and evaluate the

design in terms of its hardware module set. Using this evaluation and a set of heuristics, EXPL decides which solutions should be kept to generate other solutions by yet another application of the graph transformations.

Figure 5 is the ISP description of an 8-bit multiplier that will be used as a running example to illustrate various aspects of EXPL. The algorithm is a variant of the shift-and-add algorithm. The multiplier is in the P register and the multiplicand is in the MPD register and is assumed to occupy the leftmost 8 bits of the register. The product will be in the P register. The partial products are formed in the left hand side of the P register and shifted to their appropriated position in the final product. A counter, C, is used to keep track of the number of times the basic multiplication step has been performed. Additional details about the algorithm can be found in (Bell [72]).

The description begins with the specification of the label for the program (MULTIPLIER). Labels are used in ISP to identify activities so that they can be branched to, or used as subroutines.

```
MULTIPLIER:=
(DECLARE MPD < 15:0> ; P < 15:0> ; C < 15:0>
ERALCED
C←8 NEXT
L1:= (
(DECODE P< 0 > = > P←P ↑SR0 1;
                   P←(P+MPD) ↑SR0 1)
NEXT C← C-1NEXT
(IF C NEQ 0 = > L1)
```

Figure 5.
The ISP Description of the Multiplier.

The program itself is enclosed in parenthesis, and consists of two parts. The declarations and the specification of the behavior. The former are specified as a list of individual component declarations (multiplicand, multiplier/product, and step counter), using the reserved identifiers DECLARE and ERALCED as brackets. The specification of the activities of the system is given as a list of two sequential steps. The first step (C←8) initializes the counter and the second is given by a labeled (L1) block of activities. These consist of a sequence of three steps. The first one performs the basic multiplication operation; the second step decrements the counter; the third step tests the counter to see if the operation has been completed. If the value of the counter has not reached 0 then a jump to the label is indicated by using the label as an activity. If the counter is 0 then control flows out of the labeled statement and reaches the end of the program.

The basic multiplication operation is described using the DECODE control operation. It implements an n-way branch depending on the value of the expression following the operator. The alternative paths selected by this operation are given as a list using the ";" as delimiter. The first path (P←P ↑SRO 1) is selected if the value of the controlling expression (P < 0 >) is 0; the second path (P←(P+MPD) ↑SRO 1) is selected if the value is 1. The operator ↑SRO represents a shift right inserting zeroes. The number of shifted positions is given by the second operand (in this case the integer 1).

Figure 6 shows the graph representation of the ISP description. The mapping from the ISP description to the graph form is apparent from the example. The system graph contains a unique entry point (the START operation) and a unique exit point (the STOP operation). In addition to these two operations, there can be five other types of operations in the graph model:

- branch, activates one of the output paths depending upon the value of some operand.
- serial-merge, activates its output path when any of its input signals arrive.
- diverge, activates concurrently all of its output paths.
- parallel-merge, activates its output path when all of its input signals arrive.
- data-operation (other).

Examination of the graph for the multiplier example indicates several possible alternative designs. For instance, the computation of the loop count (C←C-1) does not depend on the shifting and adding steps (P←P ↑SRO 1 and P←(P+MPD) ↑SRO 1); the two sets of operations do not have variables in common. Thus the decrement of the

loop counter can be performed in parallel with the basic multiplication step, as shown in Figure 7.

The graph thus obtained shows that the testing of the loop counter, although independent of the multiplication steps, can not be performed in parallel with the decrement of the loop count. This fact rules out a transformation similar to the one used previously. However, it is possible to insert the testing step in the same path as the decrement. This preserves the required ordering of the counter operations but now the testing is done concurrently with the multiplication step as shown in Figure 8.

The last graph represents an "optimal" speed implementation. We are not considering, at this point, specific module-set-dependent optimizations. For instance, register allocation in the RTM data operators. This type of optimization is left up to the individual technology-dependent evaluators.

It should also be noted that, in the example above, although it took two steps to arrive at the final design we could have taken the two counter operations (decrement and testing) as one group, and place the group in parallel with the basic multiplication step. In other words, we can achieve the same final result in one step by varying the size of the graph partitions.

These graph transformations have taken us from an original solution, to two additional design alter-

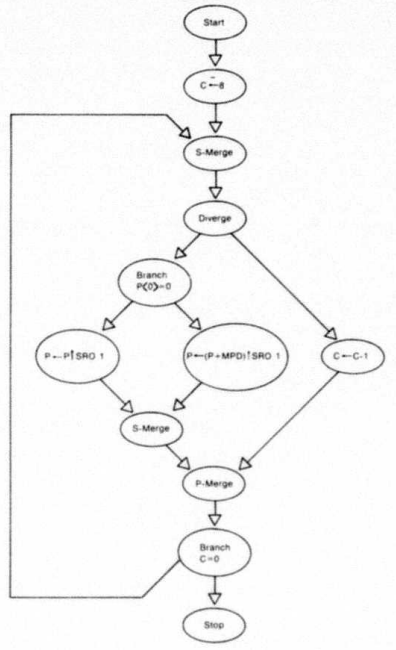


Figure 7
Multiplier Graph 1

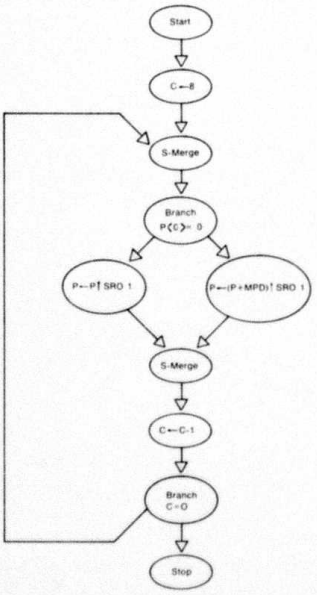


Figure 6
Multiplier Graph 0 (Original)

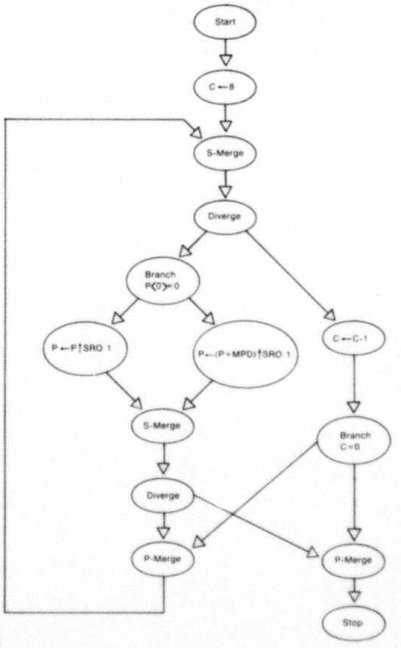


Figure 8
Multiplier Graph 2

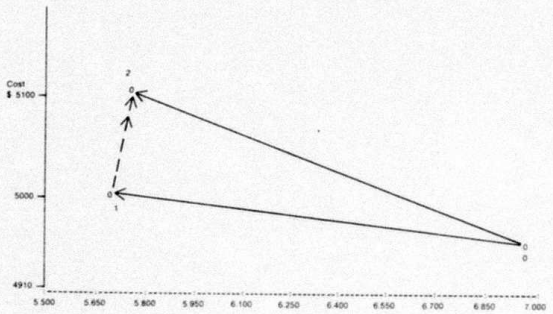


Figure 9
The Multiplier Design Space (Macromodules)

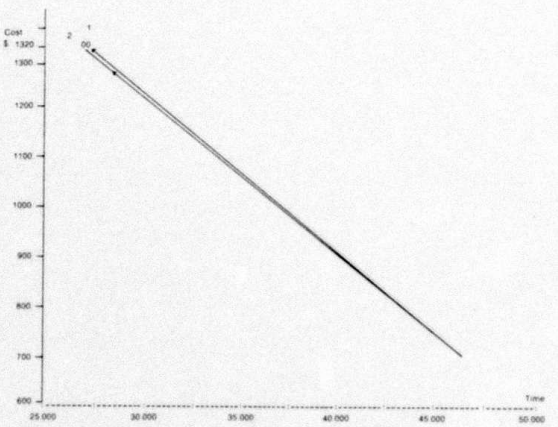


Figure 10
The Multiplier Design Space (RTM)

natives. Both represent an improvement in speed with respect to the initial design. The design space, explored automatically by EXPL, can be represented as a two dimensional plot. Each design is represented by its cost and time coordinates, computed by the technology-dependent design evaluators.

The multiplier example was implemented using both RTM's and macromodules and the design spaces are shown in Figures 9 and 10. In both plots, point 0 represents the initial solution and points 1 and 2 represent the alternative designs. Arrows indicate the steps taken in the derivation of the alternatives.

The position of the alternative design in the design space is dictated by the evaluation of its cost and speed. These two parameters are measured in terms of a specific technology, using specialized evaluator routines, as described in the following section.

IV. Module Set Evaluators

Given a candidate design, whether it is the initial graph or one obtained by a transformation, its cost and speed must be measured to ascertain its relative position in the design space. This position indicates the quality of a solution. The evaluation process is clearly dependent on the module set used and is currently performed by ad-hoc routines, independent of the graph transformation algorithms. This section describes the evaluation procedure used for RTM (Bell [72]) and macromodule (Clark [67]) systems.

The candidate design is represented by a description of its behavior, i.e., a graph. In this representation the control and data operations are not bound to specific physical components. The evaluation is performed by applying a series of binding algorithms that map this (abstract) behavioral description into a physical description. This is the representation which is then evaluated. It should be pointed out that the design space we are dealing with is really an evaluation space, not a structural space. In other words, for each point in the evaluation space there may be more than one structure.

The order in which the binding process is performed is dictated by the interconnection rules of the module set. Specifically, in the RTM set all data operators and operands are connected to a bus, Figure 11b, which not only takes part in all data operations but also has some control function. This feature of RTM requires that the binding of the buses precede the binding of the data modules. Moreover, in RTM, concurrency of operations is dictated by the number of available buses. Thus the selection of the buses must follow the binding of the control

operators. The order of bindings in RTM is the following:

(1) Selection and identification of control modules.
(2) Selection and identification of the buses required to implement the control flow structure.
(3) Selection and identification of the data modules that are to be attached to each individual bus.

Macromodules represent a different design philosophy. Modules are connected directly and there is no need for buses as in the RTM set. The basic feature of the modules is the register stack. The register or memory module occupies the base of the stack and takes part in the data operations plugged in above it, Figure 11a. Any data operation whose direct output is to be stored in register X must be physically located above register X and cannot be shared with other registers. Thus, the data operands dictate, to a large degree, the structure of the system. Special cables are used to interconnect stacks. Control signals are generated upon the completion of a data operation in order to initiate the next operation(s). The order of binding is dictated by these interconnection rules:

(1) Selection and identification of data modules.
(2) Selection and identification of the data operators that are to be attached to each data operand.
(3) Selection and identification of the control modules.

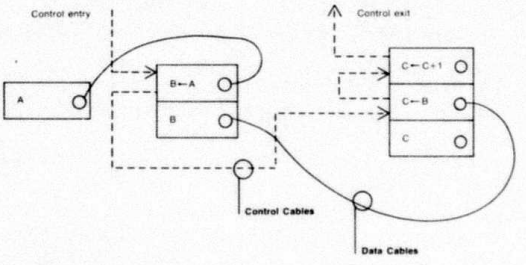
In both RTM's and macromodules the binding of control operations to control modules is usually straightforward. There is a one-to-one correspondence between the abstract control operations and the control modules.

The binding of the data operations is more difficult. In general there is not a one-to-one mapping from the operations in the graph to the operators in the module set. For instance, in RTM the data operations are performed in a general purpose arithmetic unit (the DMgpa module). Mapping from the abstract operations in the graph into the RTM structure requires the use of specialized templates. These templates act like macros in an assembly language. For each abstract data operation in the graph there is a template that indicates the number and order of RTM operations that must be executed in order to achieve the desired effect. For instance, an operation like A ← B + C is mapped into a sequence of RTM primitive operations like: DMgpa/A ← B; DMgpa/B ← C; A ← DMgpa/A + DMgpa/B.

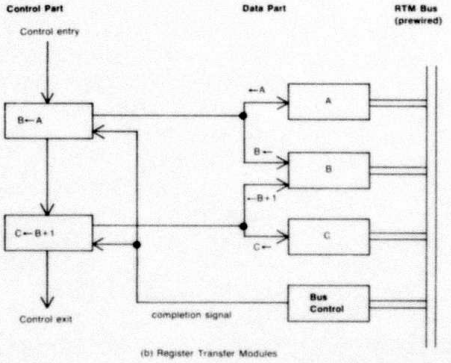
Templates of a similar nature are used by the macromodule evaluator, although in this case the nature of the module set allows more flexibility. For instance, using the example above, there are several ways of implementing the statement and the choice can be critical. Since there are no general purpose arithmetic units (the operator modules are specialized) the statement A ← B + C can be implemented alternatively as: A ← B; A ← A + C or A ← C; A ← A + B.

The decision is critical and depends upon the data operators already connected to a given register (in this case, register A) or which will be connected in order to implement subsequent operations. If the operator A ← A + C has already been placed in the stack of register A, the first option is clearly the one to adopt. On the other hand, if none of the operators exists then the template adopted depends upon the future uses of the operator, a decision based on a global analysis of the graph.

From the previous discussion, it is clear that in RTM's the critical choices are associated with the binding of the data operands to memory modules. For instance, we can allocate a variable to one of the



(a) Macromodules



(b) Register Transfer Modules

Macromodules and RTM Systems to Implement B←A next C←B+1

Figure 11

two registers of an arithmetic unit. There are certain trade-offs that can be achieved between the time taken to move variables in and out of these special registers versus the cost of adding extra arithmetic units in order to use their registers. In macromodules the critical choices are associated with the binding of data operations to data operators. For instance, we can opt to implement all data operators in terms of an auxilliary register, used as an accumulator. There are trade-offs between the time needed to route the data to and from the accumulator versus the cost of having more specialized data operators, associated with the individual registers.

An important distinction between macromodules and RTM is the degree and flexibility with which concurrency of operations can be implemented. As we mentioned before, RTM operands (registers, memories, etc.) are physically connected to a single bus. This implies that variables can not be readily shared by concurrent computations. Common variables must be copied and allocated separately—a process which degrades both the cost and speed of the design. Macromodules on the other hand allow almost unlimited concurrency; variables are accessible directly from any part of the system and there is no need to allocate extra copies. These properties of the module sets imply that, while in macromodules the intuitive feeling that parallelism implies extra cost and extra speed holds true, in RTM's the need to allocate and transfer variables between the buses may so degrade the performance that for certain systems more concurrency implies more cost and less speed.

V. Heuristics and Design Space Trade-Offs

Due to the interaction between series/parallel transformations in EXPL it is a difficult task to formalize the optimization (improvement of alternative structures) as a mathematical optimization problem. The main difficulty is the fact that transformations apply to subgraphs of arbitrary size and, as a consequence transformations in a given alternative structure may or may not be feasible or desirable in structures derived from it. It is also the case that new cases of transformations become feasible or desirable only after a specific sequence of transformations has been applied.

Two parameters will be used to describe the design space: The cost of the hardware involved and the operational time. The former is obtained by adding the costs of the components used in both the data and control structures. The latter is obtained from the average speed of the operations involved.

For a straight sequence of operations the time required is the sum of the individual times, Figure 12.a. In the presence of concurrent activities, the operation time is that of the longest (timewise) sequence, Figure 12.b.

When computing the times required by the alternative paths of a branch operation EXPL assumes, by default, that all such paths have equal probabilities of being executed (the probability being 1/n for n-way branches). This default can be overruled by the user by specifying the branching probabilities for the individual paths. The computation of the times required by the paths is then weighted by the branching probability associated with the path, Figure 12.c. The execution time is then the sum of these weighted path times.

The presence of cycles (loops) adds some complexity to the estimation of the operation time. In this case the level of nesting is assumed to be proportional to the frequency of execution of the operations. Conceptually this is equivalent to replacing the cycle by a sequence of multiple copies of the individual operations. Since the number of times a loop is executed (i.e., the number of copies) is usually unknown, a default (2) is assumed (this is a consequence of the default 50% probability of branching back to the loop head). This default may be overruled by the designer by specifying an estimate loop count or, alternatively, simply the branching probabilities if the loop count is not known, Figure 12.d.

Having defined the parameters of the design space we can now describe the trade-offs involved in the transformation rules. Connectivity and data dependency are used in the system to indicate the feasibility of a transformation. Feasible transformations, however, do not imply necessarily any advantage in their application and the desirability of such a transformation is indicated by a different set of conditions.

The exploration of the design space in our system is performed by a group of heuristic routines that produce alternative designs in a goal oriented fashion; the goal being specified by the designer. Ideally, the goal is to find an alternative structure whose position in the design space is as close as possible to the origin (0 cost and 0 time). This ideal case is, however, not easily found in real solutions. The usual case is that the least expensive solution is not the fastest and vice versa. This characteristic provides a rough classification of the design objectives into two classes: minimal cost and minimal time.

Although a designer's aim can be classified according to these objective functions, it may be the case that the real objective is more complicated in nature, namely, some combination of time and cost. For instance, the objective could be something like: "the

fastest alternative structure not costing more than x dollars."

For simplicity, the subspace of acceptable solutions will be defined by a set of straight-line segments whose slopes reflect the objective functions. In the example above a single straight line, parallel to the cost axis, would be used to divide the space in two halves. Only those solutions that lie in the semispace containing the origin are considered acceptable. These solutions represent improvements along the design goal.

More complex constraints can be described by using lines of the form $C=-m*T+b$, where m is a parameter indicating how many dollars the designer is willing to pay for each time unit saved (if time is the primary goal) or how many time units the designer is willing to sacrifice for each dollar saved (if cost is the objective). An example Figure 13, will clarify this description.

Assume that the primary objective is a reduction in time and that the designer wants a time/cost trade-off of at most m dollars for each time unit improvement. Furthermore, assume that the original design is characterized by C1 and T1. The "acceptable trade-off" subspace would thus be delineated by two line segments: one parallel to the cost axis starting from (T1,C1) to (T1,0), and the other through (T1,C1) with slope -m. By studying the control flow and data dependencies in this original structure, four transformations are available which yield four alternative solutions derived from the original one: A,B,C,D.

By dividing the space according to the trade-off lines alternatives B, C, and D can be rejected because their characteristics are not within the acceptable subspace (i.e., they take more time or the decrease in time costs too much). The alternative left, A, represents improvement in time while the cost to achieve the improvement is under the designer's threshold.

The process can now be applied to A in an identical manner. Design A is taken as the new initial solution and a new "acceptable trade-off" subspace is defined by a line segment (T2,C2) to (T2,0) and a line with slope -m through (T2,C2). Since in some cases more than one alternative can be left for further exploration, this process takes the form of a tree walk where the nodes represent alternative solutions and the edges are the transformations applied. In some instances, identical structures can be obtained by different sequences of transformations and the exploration of the design space is a graph-walking process. In any event, a path ends when no alternative solutions worth exploring can be reached from a given point. When all possible paths have been explored the end nodes are measured

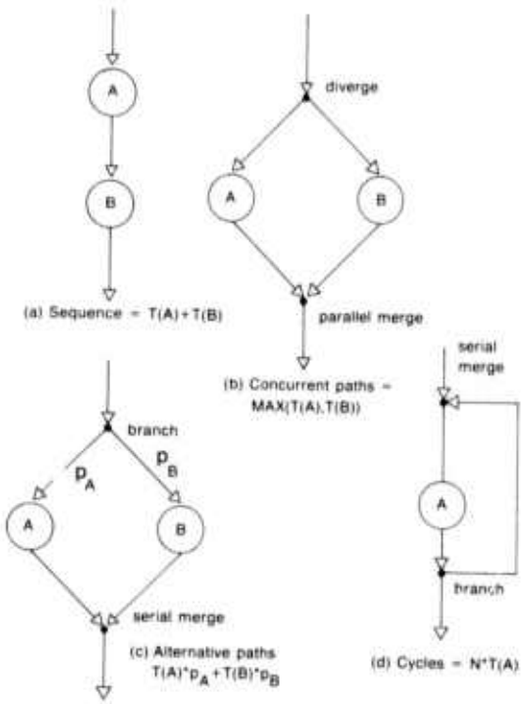


Figure 12
Time Estimation

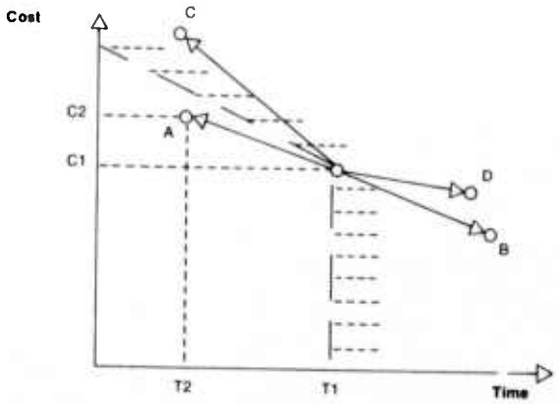


Figure 13
Design Space Reduction

against the primary objective and the best one chosen.

In general, the space of alternative solutions looks more like a graph than a tree. Several paths (i.e., sequences of transformations) may lead to the same solution. Thus, it is important to detect points in the space that have already been examined. Other problems that arise in the exploration process have to do with the cost of the process itself. EXPL does not perform a brute force search. Accepting an alternative solution for further exploration depends upon the goals indicated by the user. Besides the main goals (speed, cost, and a trade/off factor) mentioned before, the user can also specify a minimum percentage gain for a transformation-derived solution to be acceptable. If the gain falls below this threshold the new design is rejected. This pruning process, when applied indiscriminately, can lead to an incomplete exploration. It may be the case that although, a derived solution is worse (according to the goals) than its parent solution, solutions derived from the former could in fact be better than the parent. EXPL handles the detection of this type of local optimality by allowing the user to specify a rejection level. The rejection level indicates whether or not non-improving solutions are to be further explored. The user specifies the maximum length of such non-improving paths.

The following section briefly presents several examples of design spaces. The examples illustrate some of the points discussed previously.

VI. Sample Design Spaces

In this section we will present three examples of the design spaces explored by EXPL. We will not discuss the specific systems whose design spaces are depicted in Figures 14, 15, and 16. The examples will be used to show the characteristics of the design spaces and the exploration procedures.

Figure 14 shows the design space for a RTM system that is used as a controller for the X- and Y-plates of an oscilloscope. The system is used at CMU for RTM demonstrations (the "Munching Squares Generator"). The first characteristic that can be noticed is the stratification of the alternative designs. The solutions appear in horizontal bands representing solutions of similar cost. This is due to the high cost of the RTM buses compared with the cost of the other modules in the RTM set. The space is divided into bands corresponding to the 1, 2, 3, and 4 bus solutions.

The figure shows the degrading effect in RTMs of sharing variables between concurrent computations. The best solution (in terms of speed) used 3 buses and is faster than the 4 bus solutions. The algorithm is such that, although it allows a high degree of concurrency, when this degree exceeds a certain threshold there is a loss of speed in the total system. The path followed to find the best solution is indicated in the figure. It is interesting to observe the transition from solution 2 to solution 3. There is a substantial gain in speed together with a reduction in cost. The explanation is that once the cost of a bus has been accepted as a reasonable price to pay for a given gain in speed it does not cost much to spread the load and perform more operations concurrently. Indeed, as the example shows, alternative allocation of the computations to the buses, for a fixed number of buses, is crucial.

Figure 15 depicts a feature of the search procedure used in EXPL. When a solution is analyzed the set of feasible transformations that can be applied to its graph is tabulated. The improvement factor specified by the designer is then used to prune this table. This pruning takes place before a transformation is applied and is based on a preliminary "best case" analysis of a candidate transformation. The solution derived by applying the transformation may or may not realize the potential gain indicated by the preliminary analysis. This reduction in the predicted gain is due to several causes. If the goal is a reduction of cost, performing two concurrent operations in sequence may not in the case of RTMs result in a reduction in the number of buses (other computations may require the bus that was thought to be expendable). If the goal is a gain in speed, adding buses may result in a loss of speed due to the time required to copy and move variables between the buses in the system. Similar considerations can be applied to the case of macromodules.

Figures 14 and 15 correspond to the design space for the same RTM system explored using different improvement thresholds. In the space shown in Figure 15, the preliminary improvement threshold was set to a higher level (20%) than in the space shown in Figure 14 (10%). An interesting phenomenon occurred. The transformation indicated by the directed line in Figure 15 had a very promising preliminary evaluation (over 30% predicted gain). When the transformation was applied, the new solution did not realize the predicted gain. It was, nevertheless, better than the original solution and was later chosen by the system as the best solution. All feasible (i.e., applicable) transformations to this new solution were then examined and none of them promised to be better than the threshold. All of these transformations were then rejected and the exploration path was terminated. When the same situation appeared in the example of Figure 14, there were several transformations that were better than the new, lower,

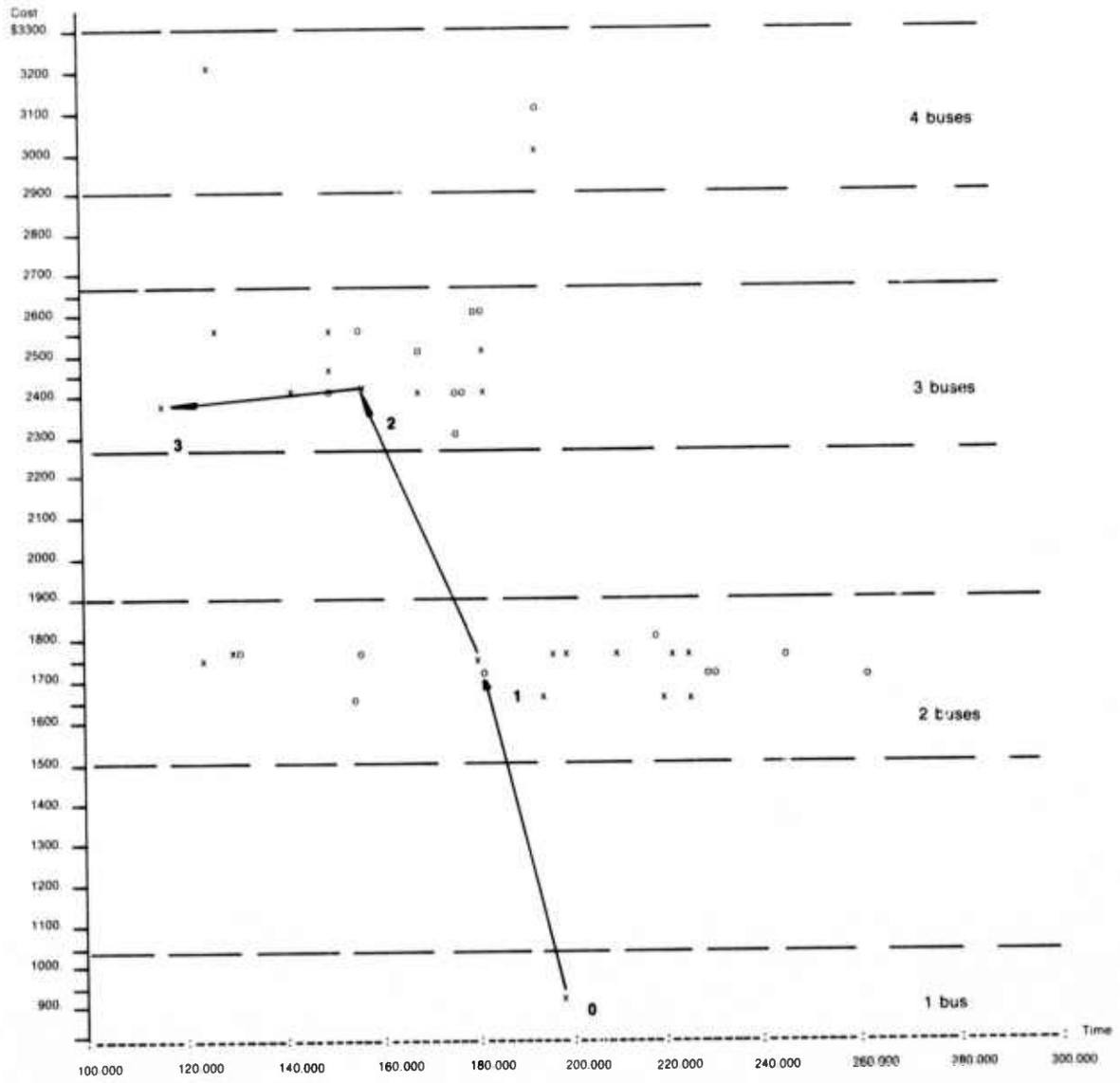


Figure 14
RTM Design Space (MSG System with 10% Improvement Factor)

threshold. One of them led in fact to the best solution of the space of Figure 14. It is interesting to observe in Figure 14 that the slope of the transformation from solution 1 to solution 2 indicates a better cost/speed trade-off than the transformation for the original solution -point 0 - to solution 1. The gain in speed produced by the transformation, although smaller than the threshold used in Figure 15, was achieved completely; there was no overhead added to the system by the extra concurrency.

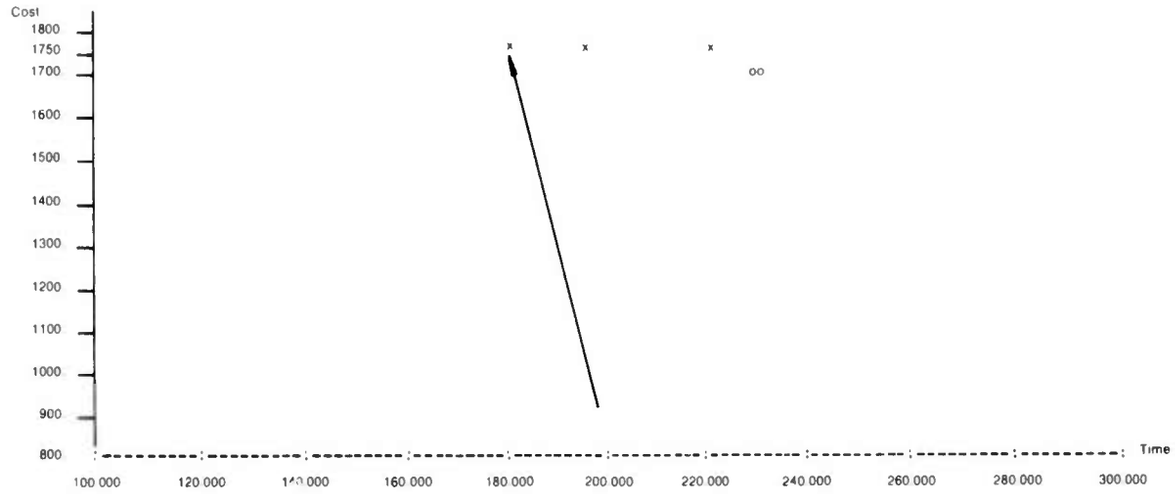


Figure 15
RTM Design Space (MSG System with 20% Improvement Factor)

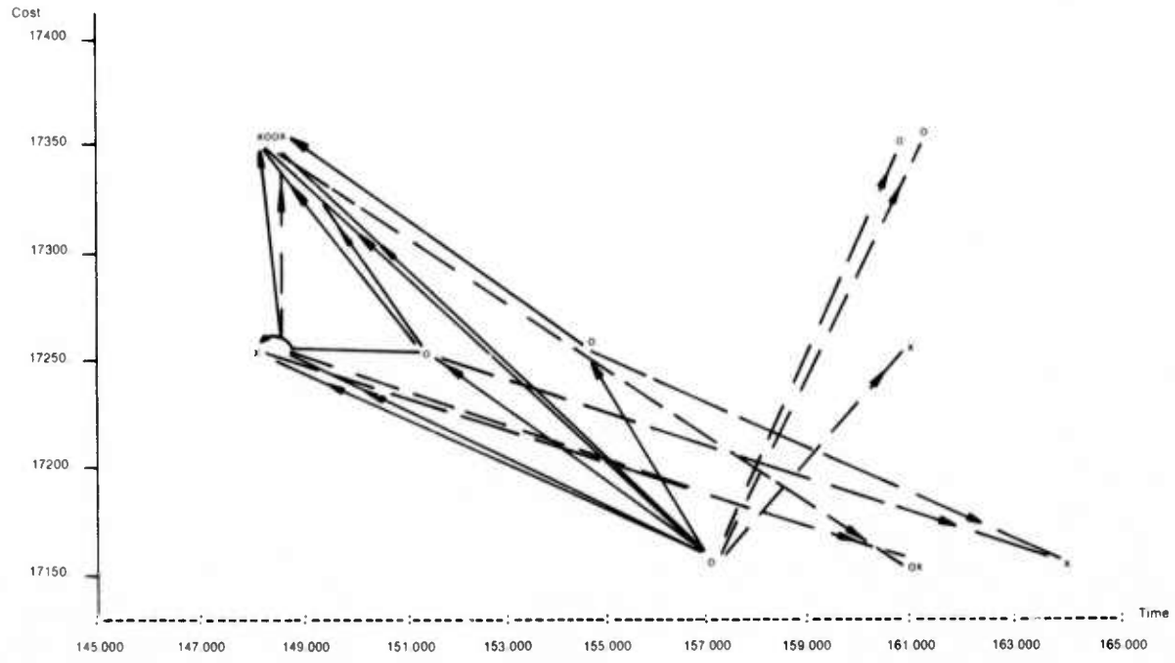


Figure 16
Macromodule Design Space (Conveyor-Bin System)

This type of anomalies is not uncommon in the modular design spaces explored so far, if anything, they tend to be the rule rather than the exception. The pruning of the applicable transformations, based on a preliminary analysis, can lead us to ignore certain transformation paths that may yield better solutions. EXPL is, in this sense, not very smart. Better heuristics are needed and research in this area is actively pursued by the implementors of the system. It is valid to ask, "why then should the system do any pruning at all?". The only reason we can provide is based on the analysis of the cases studied so far. Applying a transformation without any considerations to its possible gain is an expensive proposition. For any solution, branching factors (i.e., number of feasible transformations) of 30 to 50 are not uncommon. Applying a transformation implies a reconfiguration of the graph and the recomputation of several associated tables—an expensive operation in the current implementation. Applying each feasible transformation can lead to a very expensive design process. The system as implemented allows the designer to guide the exploration via an interactive command language. In this interactive mode, EXPL does not perform any pruning and the designer is free to order the system to perform any feasible transformation, regardless of its predicted gain. The automatic mode of exploration can therefore be used selectively under user guidance.

Figure 16 shows the design space for a system designed as a controller for a conveyor-bin unit. The design space corresponds to the alternative designs implemented using macromodules. The figure is a good example of a design space with multiple paths leading to the same solution. The space configuration also indicates the charcteristic of macromodular systems. Once a basic design is implemented, variations in the level of concurrency do not present the radical changes in cost typical of RTM system. The basic costs of the macromodular system are given by the memory and data operation modules (the "stacks"). Variations in concurrency only imply adding or eliminating control modules and cables, a minor fraction of the total cost.

VII. Other Design Tools

Another application is design verification. It is possible to develop an ISP description that is syntactically correct but that does not make sense logically. Figure 17.a depicts a syntactically correct ISP while Figure 17.b illustrates the corresponding graph. The graph is essentially the same one produced by the ISP compiler. The data operations have been deleted as a notational convenience (we can think of the data operations as being assimilated into the arcs connecting the control operations).

In the case of x = 1 the right half of the parallel merge in the graph would receive two control signals (one from the right half of the diverge, the other from the left half via the branch). The other input to the parallel merge would not receive a control signal and the system would deadlock at the parallel merge. Analytical tools based on the vector addition system (VAS) (Huen [75]) have been programmed to detect such design flaws.

The VAS is best introduced by example. Consider Figure 17.b. The arcs in the graph represent register transfers while the vertices represent control primitives. Each arc may contain tokens representing evocation of the associated register transfer. Graphically a token is represented by a dot on an arc. A marking of a graph with r arcs is a mapping from the set of r arcs to an r-dimensional vector of nonnegative integers, each of which represents the number of tokens on the corresponding arc.

A vertex with a token on its single input arc is said to be enabled. Only enabled vertices can fire. The firing of a vertex removes a token from the input arc and deposits a token on its output arc. For the case of multiple input arcs there is an associated logic condition, either disjunctive (signified by a +) or conjunctive (*). A vertex with disjunctive input arcs is

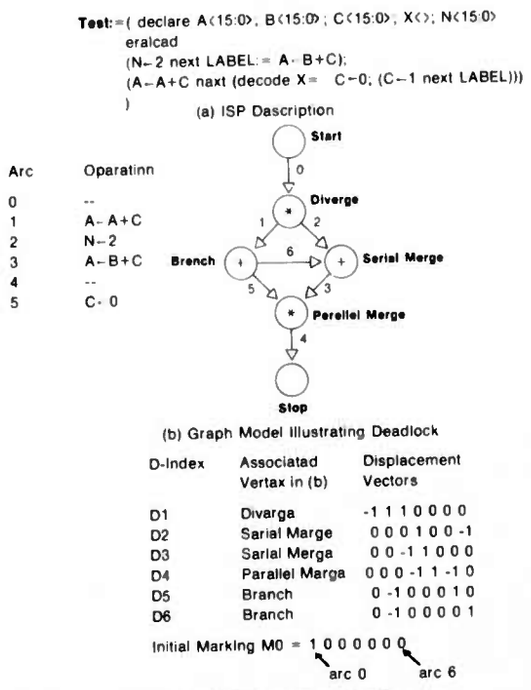


Figure 17

enabled when any input arc has a token. Firing the vertex removes a token from one input arc. This corresponds to a serial merge in the compiler produced graph. A conjunctive input condition requires tokens on all input arcs before the vertex is enabled (parallel merge). Firing the vertex removes a token from all the input arcs. Likewise a set of output arcs can be disjunctive or conjunctive. When a vertex with disjunctive output condition fires it places a token on one of the output arcs (branch). The conjunctive condition places a token on all the output arcs (diverge). A simulation is a sequence of permissible vertex firings.

Simulations are conveniently represented by the Vector Addition System (VAS) (Huen [75]). Figure 17.c depicts the VAS for the graph in Figure 17.b. The VAS consists of an initial marking vector Mo and a set of displacement vectors which correspond to vertices. Each component of the vector corresponds to an arc. All valid firings (new markings) of the graph can be determined by adding a displacement vector to the current marking Mi. Those additions which result in all marking vector components being nonnegative are valid markings and can be used to establish subsequent valid markings. For example, the only valid marking from the initial marking Mo resulting from the addition of a single displacement vector (e.g., D1) in Figure 17.c is (0,1,1,0,0,0,0). The displacement vector D2 does not lead to a valid marking since the result of its addition to Mo is (1,0,0,1,0,0,-1).

A control flow tree depicting all possible markings (or states) of the VAS can be constructed. A portion of that tree for our example is shown in Figure 18. Nodes are appended to the tree unit, for each leaf, either its marking is identical to that of one of its ancestors or no displacement vectors can be applied. In either case the node is called a leaf.

Properties of this tree can be used to detect properties of the graph. For example, the leaf (0,0,0,-0,1,0,0) represents a properly terminating sequence since there is a single token on arc 4. By contrast, leaf (0,0,0,2,0,0,0) represents two tokens on arc 3. No tokens are on the exit arc. This is the deadlock situation alluded to earlier. Furthermore, depending on the actual physical implementation of the graph, this leaf may indicate a lost signal.

Another obvious application is a simulator. The subroutine calls produced by the ISP compiler make the generation of a simulator particularly easy. Data subroutines update the data structures and control subroutines direct the flow of the simulation. A command language allows the user to direct the simulation and examine the state of various data structures. It is also desirable to produce designs according to criteria other than the traditional cost/speed criteria. One such criterion is testability. The structure of the final design substantially determines the ease with which tests can be generated for the design. A testability measure (Stephenson [74]) has been developed that correlates well with actual test generation effort. It is important to note that the common representation used as input to the various design programs is a critical feature that insures that the algorithm being evaluated is actually the one being implemented, verified, or simulated by the other design programs.

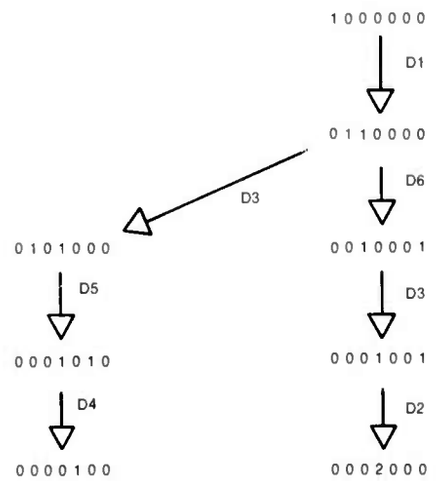


Figure 18
Portion of the Control Flow Tree for the Example of Figure 17

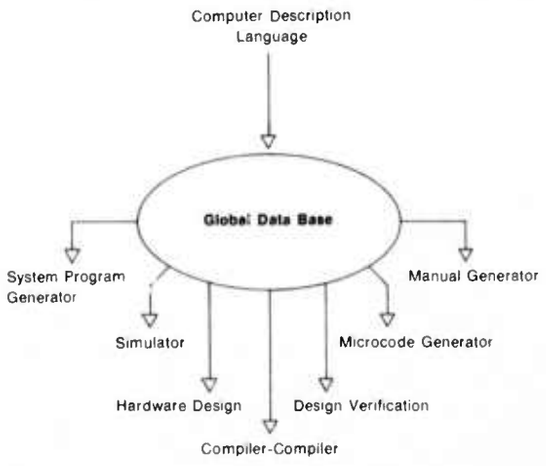


Figure 19 — SMCD
The Symbolic Manipulation of Computer Descriptions

VIII. Future Directions

To achieve the goal of automatic design relative to technology a mechanism is required that would take the description of a module set and create the equivalent of the ad hoc module set evaluators currently in use. It was also noted in section IV that the order of physical allocation (registers, buses, operators, etc.) is a strong function of the design style imposed by the module set. This information would also have to be extracted from the module set description.

The preliminary design automation system and a machine relative optimizing compiler-compiler project serve as a stepping stone to an even more ambitious project termed the Symbolic Manipulation of Computer Descriptions (SMCD) (Barbacci [74]), depicted in Figure 19. There is a continual stream of new machines spurred by the advent of minicomputers and microprocessors. Each machine has a different instruction set. The emergence of microcoded systems with the option of user-defined instruction sets has increased this flow of instruction sets. Each new system requires supporting software and the amount of software grows for any individual system as user requirements grow.

One direction in which to seek a solution to ease the burden of software development is to relativize the production of software to the description of the machine. The central ingredient of this approach is the description of computer systems in a symbolic form, such that a range of problems can be solved by manipulation of these descriptions.

Figure 19 depicts the scope of the SMCD project. The ultimate goal would be to produce and evaluate a computer system from its behavioral specifications, together with the documentation and system programs. Thus the delay from the conception of a new architecture to the time it is implemented and ready for users can be significantly reduced.

References

Barbacci [73a] Barbacci, M. R.—Automated exploration of the design space for register transfer (RT) systems. Doctoral dissertation, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., 1973.

Barbacci [73b] Barbacci, M. R. and Siewiorek, D. P.—Automated exploration of the design space for register transfer systems. *Proceedings of the First Annual Symposium on Computer Architecture*, University of Florida, Gainesville, Fl., 1973. *ACM SIGARCH, Computer News 2, 4* (1973).

Barbacci [74] Barbacci, M. R. and Siewiorek, D. P.—Some aspects of the symbolic manipulation of computer descriptions. Technical report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., 1974.

Barbacci [75] Barbacci, M. R.—A users guide to the ISP compiler. Technical report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., 1975.

Bell [71] Bell, C. G. and Newell, A.—*Computer Structures, Readings and Examples*. McGraw Hill Book Company, New York, N.Y. 1971.

Bell [72a] Bell, C. G., Eggert, J. L., Grason, J. and Williams, P.—The description and use of register transfer modules (RTM). *IEEE-TC C-21, 5* (1972), 495-500.

Bell [72b] Bell, C. G., Grason, J. and Newell, A.—*Designing Computers and Digital Systems*. Digital Press, Digital Equipment Corp., Maynard, Ma., 1972.

Breuer [72] Breuer, M. A.—Recent developments in the automated design and analysis of digital systems. Proceedings of the *IEEE 60, 1* (1972), 12-27.

Clark [67] Clark, W. A.—Macromodular computer systems. *AFIPS Conference Proceedings 30, SJCC*, Atlantic City, N.J., 1967, 335-402.

Darringer [69] Darringer, J. A.—The description, simulation, and automatic implementation of digital computer processors. Doctoral dissertation, Electrical Engineering Dept., Carnegie-Mellon University, Pittsburgh, Pa. 1969.

Friedman [69] Friedman, T. D. and Yang, S.—Methods used in an automatic logic design generator (ALERT). *IEEE-TC C-18, 7* (1969), 593-614.

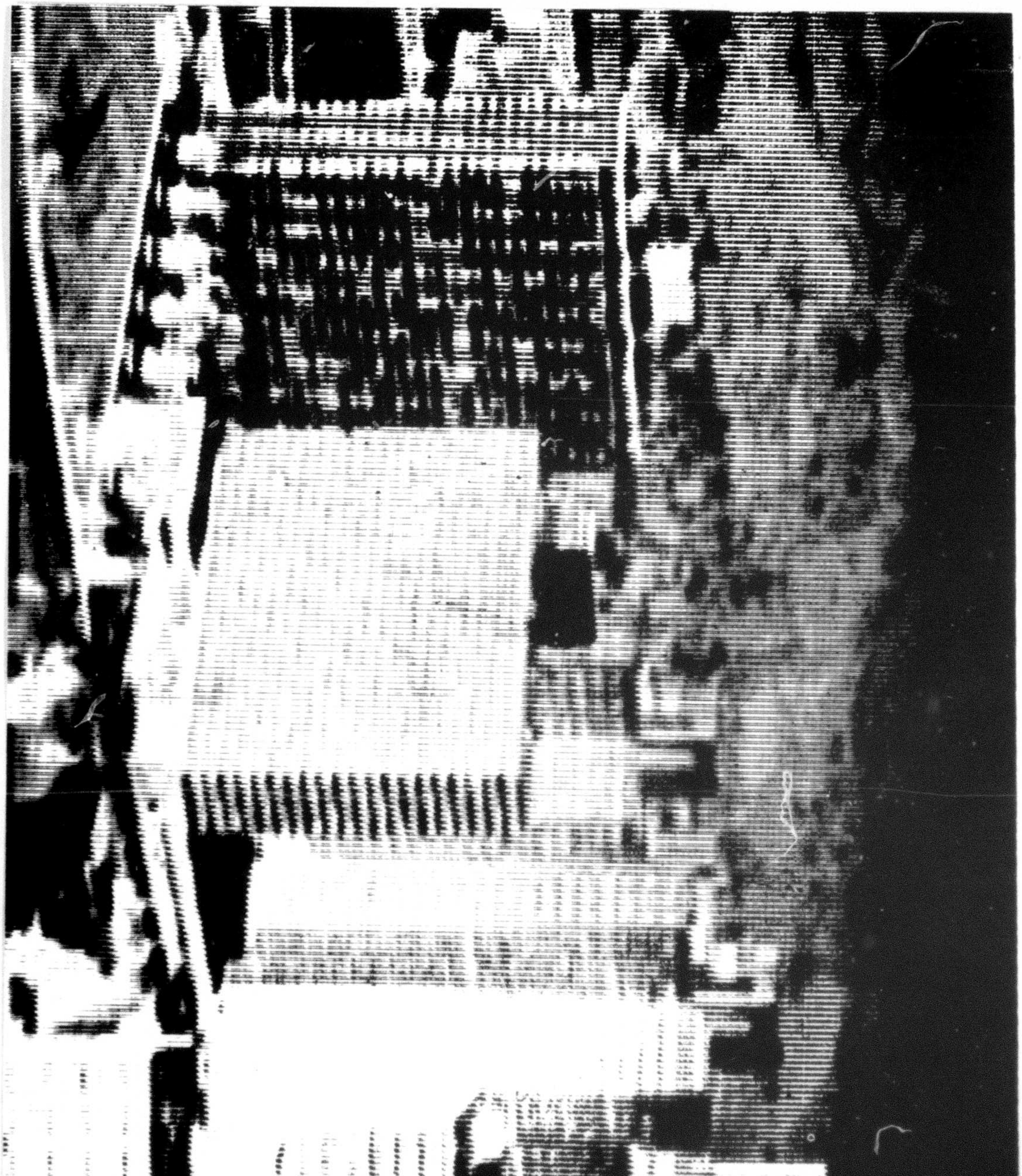
Huen [75] Huen, W. H. and Siewiorek, D. P.—Inter-module protocol for register transfer level modules: taxonomy and Analytic tools. *Proceedings of the Second Annual Symposium on Computer Architecture*, Houston, Tx., 1975.

Mesztenyi [68] Mesztenyi, C. K.—Computer design language. Simulation and Boolean translation. Technical report, 68-72, Computer Science Center, University of Maryland, College Park, Md., 1968.

Parnas [67] Parnas, D. L. and Darringer, J. A.—SODAS and a methodology for systems design. *AFIPS Conference Proceedings, FJCC*, Anaheim, Ca., 1967.

Rozenberg [71] Rozenberg, D. P. and Savage, R. L.—A proposal for the computer design process based on multi-level simulation. *IFIPS Congress*, Ljubljana, Yugoslavia, 1971.

Stephenson [74] Stephenson, J.—A testability measure for register-transfer level digital circuits. Doctoral dissertation, Electrical Engineering Dept., Carnegie-Mellon University, Pittsburgh, Pa. 1974.



Faculty and Visitors

Mario Barbacci
Research Associate
B.S., Universidad Nacional de Ingenieria, Lima, Peru (1966)
Engineer, Universidad Nacional de Ingenieria, Lima, Peru (1968)
Ph.D., Carnegie-Mellon University (1974)
Carnegie, 1969: Design Automation, Computer Architecture, Automatic Programming

C. Gordon Bell
Professor of Computer Science and Electrical Engineering
S.B., Massachusetts Institute of Technology (1956)
S.M., Massachusetts Institute of Technology (1957)
Carnegie, 1966: Computers and Computer Networks

Hans Berliner
Research Associate
B.A., George Washington Universi y (1954)
Carnegie, 1969: Artificial Intelligence

Richard P. Brent
Visiting Researcher
B.S., Monash University (1968)
M.S., Stanford University (1970)
Ph.D., Stanford University (1971)
Carnegie, 1975: Numerical Analysis, Analysis of Algorithms, Comp'tational Complexity

Jack R. Buchanan
Assistant Professor of Computer Science and Industrial Administration
B.S., University of Utah (1965)
M.A., University of Utah (1967)
Ph.D., Stanford University (1973)
Carnegie, 1972: Automatic Programming, Complex Information Processing, MIS

John Burge
Visiting Scholar
B.Sc., Sussex University (1970)
Dip. Computer Science, Cambridge University (1971)
Ph.D., Durham University (1975)
Carnegie, 1974: Artificial Intelligence

Alan Cole
Visiting Researcher
B.A., Hope College (1966)
M.A., University of Michigan (1967)
M.S., University of Michigan (1970)
Carnegie, 1975: Speech

Ludwik Czaja
Visiting Assistant Professor
M.S., University of Warsaw (1960)
Ph.D., University of Warsaw (1972)
Carnegie, 1975: Programming Languages, Semantic and Syntactic Models of Programming Languages, Compilers

Charles M. Eastman
Associate Professor of Architecture, Computer Science, and Urban and Public Affairs
B. Arch., University of California at Berkeley (1964)
M. Arch., University of California at Berkeley (1966)
Carnegie, 1967: Computer-Aided Design Cognitive Processes in Design, Urban Models

Lee D. Erman
Research Computer Scientist
B.S., University of Michigan (1966)
M.S., Stanford University (1968)
Ph.D., Stanford University (1974)
Carnegie, 1970: Artificial Intelligence, Speech Understanding

Samuel H. Fuller
Assistant Professor of Computer Science and Electrical Engineering
B.S.E., University of Michigan (1968)
M.S., Stanford University (1969)
Ph.D., Stanford University (1972)
Carnegie, 1972: Performance Evaluation, Measurement of Computer Systems, Computer Architecture

Robert G. Goodman
Visiting Scientist
B.S., Oklahoma State University (1963)
M.S., Oklahoma State University (1965)
M.S., Stanford University (1969)
Carnegie, 1974: Artificial Intelligence

A. Nico Habermann
Professor of Computer Science
B.S., Free University, Amsterdam (1953)
M.S., Free University, Amsterdam (1957)
Ph.D., Technological University, Eindhoven, The Netherlands (1967)
Carnegie, 1968: Operating Systems and Programming Languages

Louis Hageman
Senior Lecturer
B.A., DePauw University (1955)
B.S., Rose-Hulman Institute of Technology (1955)
M.S., University of Pittsburgh (1959)
Ph.D., University of Pittsburgh (1962)
Carnegie, 1973: Numerical Analysis

Frederick Hayes-Roth
Research Associate
B.A., Harvard University (1969)
M.S., University of Michigan (1972)
Ph.D., University of Michigan (1974)
Carnegie, 1974: Artificial Intelligence, Pattern Recognition, Cognitive Psychology

Laurent Hyafil
Visiting Researcher
M.S., Ecole Polytechnique (1972)
Ph.D., Universite de Paris (1974)
Carnegie, 1975: Computational Complexity

Anita K. Jones
Assistant Professor of Computer Science
B.A., Rice University (1964)
M.A., University of Texas (1966)
Ph.D., Carnegie-Mellon University (1973)
Carnegie, 1968: Programmed Systems

Boleslaw Kacewicz
Visiting Researcher
M.S., University of Warsaw (1974)
Carnegie, 1975: Computational Complexity and Numerical Mathematics

Masahiko Kida
Visiting Scholar
B.S., Waseda University (1968)
Carnegie, 1974: Multiprocessor Systems

H. T. Kung
Assistant Professor of Computer Science
B.S., National Tsing Hua University, Taiwan (1968)
Ph.D., Carnegie-Mellon University (1973)
Carnegie, 1973: Computational Complexity, Parallel Computation, Numerical Mathematics

Victor R. Lesser
Research Computer Scientist
A.B., Cornell University (1966)
M.S., Stanford University (1970)
Ph.D., Stanford University (1972)
Carnegie, 1972: Parallel System Organization for Artificial Intelligence (e.g., Speech Understanding), Computer Architecture (Micro-programming, Multiprocessor systems), Operating Systems and Problem Decomposition for Multiprocessors

John W. McCredie
Lecturer in Computer Science
Head of Computation Center
B.E., Yale University (1962)
M.S.E.E., Yale University (1964)
Ph.D., Carnegie-Mellon University (1972)
Carnegie, 1968: Analytical Modeling, Simulation, and System Performance Evaluation

Robert Meersman
Visiting Researcher
Ph.D., Vrije Universiteit Brussel (1975)
Carnegie, 1975: Parallel Computation

John McDermott
Visiting Research Associate
B.A., St. Louis University (1966)
M.A., St. Louis University (1967)
Ph.D., University of Notre Dame (1969)
Carnegie, 1974: Artificial Intelligence, Production Systems

James Moore
Research Associate
B.S., Massachusetts Institute of Technology (1964)
Ph.D., Carnegie-Mellon University (1971)
Carnegie, 1971: Artificial Intelligence and Semantic Nets

Ian Munro
Visiting Researcher
B.A., University of New Brunswick (1968)
M.Sc., University of British Columbia (1969)
Ph.D., University of Toronto (1971)
Carnegie, 1975: Computational Complexity

Joseph Newcomer
Research Associate
B.A., St. Vincent College (1967)
Ph.D., Carnegie-Mellon University (1975)
Carnegie, 1975: Operating Systems, Programming Languages

Allen Newell
University Professor
B.S., Stanford University (1949)
Ph.D., Carnegie Institute of Technology (1957)
Carnegie, 1961: Artificial Intelligence, Psychology of Human Thinking, Programming Systems, and Computer Structures

D. Raj Reddy
Professor of Computer Science
B.E., University of Madras (1958)
M. Tech., University of New South Wales (1961)
M.S., Stanford University (1964)
Ph.D., Stanford University (1966)
Carnegie, 1969: Artificial Intelligence, Computer Graphics, and Man-Machine Communications

Mario Schkolnick
Assistant Professor of Computer Science
Electrical Engineer, University of Chile (1965)
M.S., University of California (1967)
Ph.D., University of California at Berkeley (1969)
Carnegie, 1973: Data Base Design, Complexity Theory

Daniel Serain
Visiting Scholar
M.S., L'Universite de Grenoble (1972)
Carnegie, 1974: Multiprocessor Structure, Operating Systems

Mary Shaw
Assistant Professor of Computer Science
B.A., Rice University (1965)
Ph.D., Carnegie-Mellon University (1972)
Carnegie, 1971: Programming Systems, Software Tools, the Programming Environment, and Concrete Computational Complexity

Linda Shockey
Research Associate
B.S., Ohio State University (1967)
Ph.D., Ohio State University (1973)
Carnegie, 1972: Linguistics and Automatic Speech Recognition

Daniel P. Siewiorek
Assistant Professor of Computer Science and Electrical Engineering
B.S., University of Michigan (1968)
M.S., Stanford University (1969)
Ph.D., Stanford University (1972)
Carnegie, 1972: Computer Architecture Automatic Design Exploration, Computer Descriptive Languages, Modeling, Fault Tolerant Computer Design

Herbert A. Simon
Richard King Mellon Professor of Computer Science and Psychology
A.B., University of Chicago (1936)
Ph.D., University of Chicago (1943)
D.Sc. (Hon.), Case Institute of Technology (1963),
D.Sc. (Hon.), Yale University (1963)
LL.D. (Hon.), University of Chicago (1964)
LL.D. (Hon.), McGill University (1970)
Fil.D. (Hon.), University of Lund, Sweden (1968)
D.Econ.Sci., Erasmus University of Rotterdam (1973)
Carnegie, 1949: Computer Simulation of Cognitive Processes, Artificial Intelligence, and Management Science

Shigeharu Sugita
Visiting Researcher
Ph.D., Kyoto University (1968)
Carnegie, 1975: Artificial Intelligence

Yoshiro Tochio
Visiting Scholar
B.S., Osaka University, Japan (1969)
Carnegie, 1974: Artificial Intelligence

Joseph F. Traub
Professor of Computer Science and Mathematics, and Head of the Department of Computer Science
B.S., City College of New York (1954)
Ph.D., Columbia University (1959)
Carnegie, 1971: Numerical Mathematics, Computational Complexity, Parallel Computation, Algorithmic Analysis

Henryk Woźniakowski
Visiting Assistant Professor
M.S., University of Warsaw (1969)
Ph.D., University of Warsaw (1972)
Carnegie, 1973: Numerical Mathematics, Computational Complexity

William A. Wulf
Associate Professor of Computer Science
B.S., University of Illinois (1961)
M.S.E.E., University of Illinois (1963)
D.Sc., University of Virginia (1968)
Carnegie, 1968: Programming Systems: Compiler Optimization, Operating Systems, Systems Programming Languages, and Multiprocessor Systems

Departmental Staff

Engineering
Mark Adam—Technician
William Broadley—Manager of Engineering Design
Paolo Coraluppi—Research Engineer
Mike Keegan—Draftsman
Tim Kirby—Staff Engineer
Stan Kriz—Engineer
Rich Lang—Technician
Mike Powell—Engineer
Brian Rosen—Staff Engineer
Ken Stupak—Technician
Jim Teter—Manager of Engineering Production
Nancy Whitaker—Technical Clerk

Office Staff
Nancy Barron—Secretary to Department Head
Mildred Black—Secretary to Professor Newell
Judith Brantley—Secretary to Business Manager
Beverly Howell—Secretary to Dr. Reddy
Dorothy Josephson—Faculty Secretary
Deborah Lemmon—Department Secretary
Paul Newbury —Business Manager
Ruth Ann Seilhamer—Assistant Business Manager
Susan Sevigny—Documentation Librarian

Programming and Operations
Patrick Banwell—Research Programmer
Christopher Cooper—Programmer
Robert Cronk—Research Programmer
Gregory Gill—Visiting Research Associate
John Godfrey—Programmer
Ralph Guggenheim—Technician/Research Assistant
Hank Mashburn—Senior Systems Analyst
Eric Ostrom—Programmer
Chuck Pierson—Research Programmer
Brian Reid—Research Programmer
George Robertson—Senior Research Programmer
Jim Skees—Operator
Howard Susman—Research Programmer
Harold Van Zoeren—Senior Research Programmer
Dave Vavra—Operator/Programmer
Howard Wactlar—Manager of Programming and Operations

Graduate Students

Guy T. Almes
B.A., Rice University (1972)
Mathematics and Electrical Engineering
M.S., Rice University (1972)
Electrical Engineering

Gideon Ariely
B.A., Hebrew University (1969)
Mathematics, Philosophy, Computer Science

Gérard M. Baudet
Diplôme d'Ingénieur, Ecole Polytechnique (1970)
Mathematics
Diplôme d'Etudes Approfondies, Université Paris IV (1971)
Computer Science
Doctorat de 3ème cycle, Université Paris VI (1973)
Computer Science

Madeline Bauer
A.B., Cornell University (1968)
Mathematics
M.A., University of Michigan (1970)
Computing and Communications Sciences

Andrew P. Buchalter
B.S., Yale University (1974)
Physics

Roderic G. Cattell
B.S., University of Illinois (1974)
Computer Science

Robert J. Chansler, Jr.
B.S., California Institute of Technology (1974)
Mathematics-IS

Douglas W. Clark
B.S., Yale University (1972)
Engineering and Applied Science

Donald N. Cohen
B.S., Carnegie-Mellon University (1973)
Mathematics

Ellis Cohen
B.S., Drexel Institute of Technology (1970)
Mathematics

Lee W. Cooprider
B.A., Oberlin College (1969)
Mathematics

William M. Corwin
B.S., Carnegie-Mellon University (1972)
Physics

Achim Eckert
Diplom Ingenieur, Technische Universität Berlin (1974)
Electrical Engineering

David C. Eklund
B.A., Harvard University (1968)
Applied Mathematics

Craig F. Everhart
B.A., Wesleyan University (1974)
Physics

Peter Feiler
Abitur, Gymnasium Bad Toelz (1971)
Mathematics
Vordiplom, Technical University of Munich (1973)
Computer Science

Richard Fennell
B.S., Rensselaer Polytechnic Institute (1969)
Physics

Lawrence E. Flon
B.S., SUNY at Stony Brook (1972)
Physics

Charles L. Forgy
B.S., University of Texas at Arlington (1972)
Mathematics

John G. Gaschnig
B.S.E.E., Massachusetts Institute of Technology (1972)
Computer Science

Henry Goldberg
S.B., Massachusetts Institute of Technology (1968)
Mathematics

Richard H. Gumpertz
S.B.E.E., Massachusetts Institute of Technology (1973)
Electrical Engineering

Samuel P. Harbison III
A.B., Princeton University (1974)
Mathematics

Don Heller
B.S., Carnegie-Mellon University (1971)
Mathematics

Paul N. Hilfinger
A.B., Princeton University (1973)
Mathematics

Steven O. Hobbs
A.B., Dartmouth College (1969)
Mathematics
B.A., University of Michigan (1972)
Mathematics (Computer Science Option)

David R. Jefferson
B.S., Yale University (1970)
Mathematics

Richard Johnsson
B.E., Vanderbilt University (1970)
Electrical Engineering

Philip Karlton
B.A., University of California, Santa Barbara (1971)
Mathematics

John R. Kender
B.S., University of Detroit (1970)
Mathematics
M.S., University of Michigan (1972)
Mathematics

Paul J. Knueven
Sc.B., Brown University (1969)
Applied Mathematics

Donald W. Kosy
B.S., University of Michigan (1967)
Science Engineering
M.S., Stanford University (1968)
Electrical Engineering
M.S., Stanford University (1969)
Computer Science

David A. Lamb
B.S., University of Waterloo (1974)
Computer Science

Bruce W. Leverett
A.B., Harvard University (1973)
Physics and Chemistry

Roy Levin
B.S., Yale University (1970)
Mathematics

Bruce Lowerre
B.S., Case Institute of Technology (1965)
Chemistry
B.S., Case Western Reserve (1970)
Mathematics

Madhav Marathe
B.S., University of Bombay (1971)
Physics
M.S., Indian Institute of Technology, Kanpur (1972)
Physics

Karla F. Martin
B.A., Western Washington State College (1967)
Mathematics, Physics
M.A., University of Oregon (1969)
Mathematics
M.A., University of Oregon (1972)
Computer Science

Philip H. Mason
B.S., Carnegie-Mellon University (1967)
Mathematics

Donald L. McCracken
B.S., Carnegie-Mellon University (1968)
Mathematics

Patrick F. McGehearty
B.A., University of Texas at Austin (1972)
Mathematics and Computer Science
M.A., University of Texas at Austin (1974)
Computer Science

Rajan S. Modi
B.Tech., Indian Institute of Technology (1974)
Electrical Engineering

David J. Mostow
A.B., Harvard University (1974)
Applied Mathematics

Joseph Newcomer
B.A., St. Vincent College (1967)
Mathematics

John D. Oakley
B.S., Harvey Mudd College (1970)
Physics
M.S., University of Wisconsin (1972)
Computer Science

Ronald Ohlander
B.S., St. Mary's College (1962)
Psychology

Crispin S. Perdue
A.B., Princeton University (1973)
Independent Program

Frederick Pollack
B.S., University of Florida (1970)
Mathematics

Keith Price
B.S., Massachusetts Institute of Technology (1971)
Electrical Engineering

Kamesh Ramakrishna
B.S., Indian Institute of Technology, Kanpur (1974)
Electrical Engineering

Elaine Rich
A.B., Brown University (1972)
Formal Language Theory

George Rolf
B.S., University of Nymejen (1966)
Mathematics
M.S., University of Nymejen (1970)
Numerical Analysis

Steven M. Rubin
B.S., Carnegie-Mellon University (1974)
Mathematics

Michael Rychener
A.B., Oberlin College (1969)
Mathematics
M.S., Stanford University (1971)
Computer Science

Steven Saunders
S.B., Massachusetts Institute of Technology (1972)
Computer Science

Edward Schneider
B.S., Carnegie-Mellon University (1970)
Mathematics

Robert W. Schwanke
B.S., Carnegie-Mellon University (1974)
Mathematics and Computer Science

Richard Smith
B.S., Houghton College (1971)
Physics and Mathematics

David K. Stevenson
B.A., Wesleyan University (1969)
English and Mathematics
M.A., University of Oregon (1972)
Mathematics

Mark Stickel
B.S., University of Washington (1969)
Mathematics
M.S., University of Washington (1971)
Computer Science

Richard J. Swan
B.A., University of Essex (1972)
Computing Science

Walter F. Tichy
Reifezeugnis, Karlsgymnasium Bad Reichenhall (1971)
Diplom-Vorprufung, Technical University Munich (1973)
Mathematics and Computer Science

Bruce W. Weide
B.S., University of Toledo (1974)
Electrical Engineering

Charles Weinstock
B.S., Carnegie-Mellon University (1970)
Mathematics

Publications

July 1, 1974 to June 30, 1975

These publications are given in alphabetical order according to the name of the first author listed for each publication. In cases of multiple authorship where more than one author is in the Computer Science Department, a cross-reference is made to that first listing under the name of each departmental author.

No cross-references are made for non-departmental authors.

Barbacci, M. R. and D. P. Siewiorek, "Some Aspects of the Symbolic Manipulation of Computer Descriptions", *Second Annual Workshop on Computer Hardware Description Languages*, Technische Hochschule, Darmstadt, West Germany, July 1974.

Barbacci, M. R. and D. P. Siewiorek, "Some Observations on Modular Design Technology and the Use of Microprogramming", *Infotech State of the Art Report on Microprogramming and Systems Architecture*, Berkshire, UK, (to appear in 1975).

Barbacci, M. R., "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems", *IEEE Transactions on Computers*, Vol. c-24, No. 2, February 1975, 137-150. PB 221591

Berliner, H. J., "A Representation of Some Mechanisms for a Problem Solving Chess Program", to appear in *Recent Advances in Computer Chess*, Edinburgh University Press.

For references by W. Broadley, see D. R. Reddy.

Cohen, E. S., "A Semantic Model for Parallel Systems with Scheduling", *Second ACM Symposium on Principles of Programming Languages*, Palo Alto, Ca., January 1975.

Cooprider, L. W., F. Heymans, R. J. Courtois and D. L. Parnas, "Information Streams Sharing a Finite Buffer: Other Solutions", *Information Processing Letters*, 3:1 July 1974, 16-21.

Eastman, C. M., J. Lividini and D. Stoker, "A Database for Designing Large Physical Systems", *1975 National Computer Conference Proceedings*, Anaheim, Ca., 1975.

Eastman, C. M. and J. Lividini, "System Design for a Building Description System", *C1B W52 Symposium on Computer Languages in Building*, Budapest, Hungary, April 1975.

Eastman, C. M., J. Lividini and D. Stoker, "A Data Structure for Building Elements", *C1B W52 Symposium on Computer Languages in Building*, Budapest, Hungary, April 1975.

Eastman, C. M. and J. Lividini, "Spatial Search" (revised), *Institute of Physical Planning*, Research Report No. 55, CMU, April 1975.

Fuller, S. H., V. R. Lesser, C. G. Bell and C. Kaman, "Microprogramming and Its Relation to Emulation and Technology", *Seventh Annual Microprogramming Conference*, Palo Alto, Ca., October 1974.

For other references by S. H. Fuller, see M. V. Marathe.

Gaschnig, J. G., "A Constraint Satisfaction Method for Inference Making", *Proc. Twelfth Annual Allerton Conference on Circuit and System Theory*, University of Illinois at Urbana-Champaign, October 1974.

Gilmartin, K. J., A. Newell and H. A. Simon, "A Program Modeling Short-Term Memory Under Strategy Control", *Psychology Dept.*, CIP Working Paper No. 293, CMU, March 1975.

Godfrey, J. D., J. M. Powell, and E. A. Snow, "A Cardiac Arrhythmia Monitoring System", I.E.E.E. Computer Society Conference, Washington, D.C., September 1974.

Grason, J. and D. P. Siewiorek, "A Modular Approach to Prototype System Construction in Real-Time Minicomputer Laboratory", *COMP CON 74, Ninth Annual IEEE Computer Society International Conference*, Washington, D.C., September 1974, 139-143.

Hayes-Roth, B. and F. Hayes-Roth, "Plasticity in Memorial Networks", *Journal of Verbal Learning and Verbal Behavior*, (to appear)

Hayes-Roth, F., "Schematic Classification Problems and Their Solution", *Pattern Recognition*, 1974, 6, 105-114.

Hayes-Roth, F., "Representation of Structured Events and Efficient Procedures for Their Recognition", *Pattern Recognition*, (to appear).

Hayes-Roth, F., "An Optimal Network Representation and Other Mechanisms for the Recognition of Structured Events", *Proc. Second International Joint Conference on Pattern Recognition*, 1974.

Hayes-Roth, F. and D. J. Mostow, "An Automatically Compilable Recognition Network for Structured Patterns", *Proc. Fourth International Joint Conference on Artificial Intelligence*, (to appear).

Heller, D., "A Determinant Theorem with Applications to Parallel Algorithms", *SIAM J. Num. Anal.*, Vol. II, No. 3, June 1974, 559-568.

Heller, D., "On the Efficient Computation of Recurrence Relations", *The Institute for Computer Applications in Science and Engineering*, NASA Langley Research Center, Hampton, Va., June 1974.

Huen, W. H. and D. P. Siewiorek, "Intermodule Protocol for Register Transfer Level Modules: Taxonomy and Analytic Tools", *Proc. Second Annual Symposium on Computer Architecture*, Houston, Tx., February 1975.

Hyafil, L. and H. T. Kung, "The Complexity of Parallel Evaluation of Linear Recurrences", *Proc. Seventh Annual ACM Symposium on Theory of Computing*, May 1975.

Jenkins, M. A. and J. F. Traub, "Principles for Testing Polynomial Zerofinding Programs", *ACM Trans. on Mathematical Software 1*, 1975, 26-34.

Jones, A. K. and W. A. Wulf, "Towards the Design of Secure Systems", *Proc. of the International Workshop on Protection in Operating Systems*, IRIA, Rocquencourt, France, August 1974, 121-135.

Knueven, P., "The Foundation of a Flexible Run-Time System for Algol 68S", *Proc. International Conference on Experience with Algol 68*, University of Liverpool Press, Liverpool, UK, April 1975.

Kung, H. T., "The Computational Complexity of Algebraic Numbers", *SIAM J. Num.* Anal. 12, 1975, 89-96.

Kung, H. T., "On the Computational Complexity of Finding the Maxima of a Set of Vectors", *Proc. Fifth Annual IEEE Symposium on Switching and Automata Theory*, 1974, 117-121.

For other references by H. T. Kung, see L. Hyafil.

Lesser, V. R., "HSII: A Multiprocess Multiprocessor Speech Understanding System", *Interface Workshop on Interprocess Communication*, 1975.

For other references by V. R. Lesser, see S. H. Fuller.

Marathe, M. V. and S. H. Fuller, "Hardware Aids to Performance Evaluation", *Proc. Tenth Annual Convention of the Computer Society of India*, January 1975.

Newell, A. and G. Robertson, "Some Issues in Programming Multi-Mini-Processor", *1974 Conference on the On-Line Use of Computers in Psychology*, Journal of Behavior Research Methods and Instrumentation, Psychonomic Society, Inc., Austin (in press).

Newell, A., "A Tutorial on Speech Understanding Systems", *Invited Papers of IEEE Symposium on Speech Understanding*, Academic Press, NY (in press).

For other references by A. Newell, see H. A. Simon.

Parnas, D. L. and D. P. Siewiorek, "Use of the Concept of Transparency in the Design of Hierarchcially Structured Systems", *Communications of the ACM*,Vol. 18, No. 5, May 1975.

Reddy, D. R., "Computer as a Research Tool in Speech Understanding Research", *Fed. of Am. Soc. for Exp. Biology*, Vol. 33, No. 12, 1974, 2347-2351.

Reddy, D. R., B. Rosen, S. Kriz and W. Broadley, "Computer Graphics in Research: Some State-of-the-Art Systems", *American Psychologist*, Vol. 30, No. 3, 1975, 239-246.

For other references by D. R. Reddy, see L. Shockey.

Saunders, S. E., "Improved FM Audio Synthesis Methods for Real-Time Digital Music Generation", *ACM Computer Science Conference 75*, 1975, (abstract).

Schkolnick, M., "The Equivalence of Reducing Transition Languages and Deterministic Languages", *Communication ACM*, September 1974, 517-519.

Schkolnick, M., "Secondary Index Optimization", *Proc. ACM-SIGMOD International Conference on Management of Data*, San Jose, Ca., May 1975.

Schkolnick, M., "The Optimal Selection of Secondary Indices for Files", *Proc. International Computing Symposium 1975*, Juan-Les-Puines, France, June 1975.

Schkolnick, M., "On a Covering Problem for Partially Specified Switching Functions", *IEEE Transactions on Electronic Computers*, (to appear).

Shaw, M. and J. F. Traub, "On the Number of Multiplications for the Evaluation of a Polynomial and Some of Its Derivatives", *Journal of the ACM 21*, 1974, 161-167.

Shockey, L. and D. R. Reddy, "Transcription of Unfamiliar Language Material", Paper presented at the *87th Meeting, Acoustical Society of America*, NY, 1974, (abstract).

Siewiorek, D. P., "Modularity and Multi-Microprocessor Structures", *Proc. Seventh Annual Workshop on Microprogramming*, Palo Alto, Ca., October 1974, 186-193.

Siewiorek, D. P., "Introducing ISP", *Computer*, Vol. 7, No. 12, December 1974, 39-41.

Siewiorek, D. P., "Introducing PMS", *Computer*, Vol. 7, No. 12, December 1974, 42-44.

Siewiorek, D. P., "Process Coordination in Multi-Microprocessor Systems", *Proc. of Workshop on the Microarchitecture of Computer Systems*, Nice, France, June 1975.

Siewiorek, D. P., "Reliability Modeling of Compensating Module Failures in Majority Voted Redundancy", *IEEE Transactions of Computers*, Vol. c-24, No. 5, May 1975, 525-533.

For other references by D. P. Siewiorek, see M. R. Barbacci, J. Grason, W. H. Huen, D. L. Parnas and H. S. Stone.

Simon, H. A. and A. Newell, "Thinking Processes", D. H. Krantz, R. C. Atkinson, R. D. Luce and P. Suppes, (ed.), *Contemporary Developments in Mathematical Psychology*, 1, San Francisco, 1974, 101-104.

Stone, H. S. and D. P. Siewiorek, "Introduction to Computer Organization and Data Structures: PDP-11 Edition", *McGraw-Hill*, April 1975, 363.

Traub, J. F., "An Introduction to Some Current Research in Numerical Computational Complexity", *Proc. of the Influence of Computing on Mathematical Research and Education Cont.*, American Mathematical Society, 1974, 47-55.

Traub, J. F., "Parallel Algorithms and Parallel Computational Complexity", *Proc. IFIP Congress*, 1974, 685-687.

Traub, J. F. and H. Woźniakowski, "Strict Lower and Upper Bounds on Iterative Complexity", *Analytic Computational Complexity*, Academic Press, 1975 (to appear)

Traub, J. F. (ed.), "Analytic Computational Complexity", *Academic Press*, 1975.

For other references by J. F. Traub, see M. A. Jenkins and M. Shaw.

For references by H. Woźniakowski, see J. F. Traub.

Wulf, W. A., "More Cost Effective Computing Through Minicomputers", *SIAM News*, Vol. 7, No. 1, February 1975.

Wulf, W. A. and R. Levin, "A Local Network", *Datamation, 21, 2*, February 1975.

Wulf, W. A., "Reliable Hardware/Software Architecture", *International Conference on Reliable Software*, Los Angeles, Ca., April 1975.

Wulf, W. A., R. K. Johnsson, C. B. Weinstock, S. O. Hobbs and C. M. Geschke, "The Design of an Optimizing Compiler", *American Elsevier*, NY, 1975.

For other references by W. A. Wulf, see A. K. Jones.

Research Reports

July 1, 1974 to June 30, 1975

These reports are registered with the Defense Documentation Center. Accession numbers assigned as of July 75, are listed after the report titles.

In cases of multiple authorship where more than one author is a Faculty member or Research Associate, a cross-reference is made to the listing under the name of the principal author.

No cross-references are made for non-departmental authors.

Ariely, G., "Verification of System Programs", Ph.D. Dissertation, 1975.

Baker, J. K., "Stochastic Modeling as a Means of Automatic Speech Recognition", Ph.D. Dissertation, April 1975.

Baker, J. M., "A New Time-Domain Analysis of Human Speech and Other Complex Waveforms", Ph.D. Dissertation, May 1975.

Barbacci, M. R. and D. P. Siewiorek, "Some Aspects of the Symbolic Manipulation of Computer Descriptions", July 1974. AD A004092

For other references by M. R. Barbacci, see P L. Karlton and D. P. Siewiorek.

Baudet, G. and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers", May 1975.

Berliner, H. J., "A Representation and Some Mechanisms for a Problem Solving Chess Program", May 1975.

Bihary, D., "Graphic Display System Monitor Manual", July 1974. AD 785410

Brent, R., "Efficient Methods for Finding Zeros of Functions Whose Derivatives Are Easy to Evaluate", December 1974.

Brent, R., "A Class of Optimal-Order Zero-Finding Methods Using Derivative Evaluations", June 1975.

Brent, R. and H. T. Kung., "$0((n \log n)^{2/3})$ Algorithms for Composition and Reversion of Power Series", June 1975.

Brooks, R., "A Model of Human Cognitive Behavior in Writing Code for Computer Programs", Volume I and II, Ph.D. Dissertation, May 1975.

For references by J. Buchanan, see M.Shaw.

Chang, H., "Analysis of Deadlock Avoidance Schemes and Resource Utilization for Non-Preemptible Resources". Ph.D. Dissertation, June 1975.

Cohen, E. S., "Semantic Models for Parallel Systems", 1975.

Erman, L. D. and V. R. Lesser, "A Multi-Level Organization for Problem Solving Using Many Diverse, Cooperating Sources of Knowledge", March 1975.

Feldstein, A. and J. F. Traub, "Order of Vector Recurrences with Applications to Nonlinear Iteration, Parallel Algorithms, and the Power Method", January 1975.

Fennell, R. D., "Multiprocess Software Architecture for AI Problem Solving", Ph.D. Dissertation, May 1975.

Fennell, R. D. and V. R. Lesser, "Parallelism in AI Problem Solving: A Case Study of HEARSAY II", May 1975.

Flon, L., "Program Design with Abstract Data Types", June 1975.

Fuller, S. H., V. R. Lesser, C. G. Bell and C. Kaman, "Microprogramming and Its Relation to Emulation and Technology", July 1974. AD 784822

For other references by S. H. Fuller, see R. J. Swan.

Gerritsen, R., "Understanding Data Structures", Ph.D. Dissertation, February 1975.

Habermann, A. N., "The Correctness Proof of a Quadratic-Hash Algorithm", March 1975.

Habermann, A. N., "Path Expressions", June 1975.

Hayes-Roth, F., "Uniform Representation of Structured Patterns and an Algorithm for Grammatical Inference", January 1975.

Hedrick, C. L., "A Computer Program to Learn Production Systems Using a Semantic Net", Ph.D. Dissertation, July 1974. AD A009142

Heller, D. E., "On the Efficient Computation of Recurrence Relations", June 1974. AD A002248

Heller, D. E., D. K. Stevenson and J. F. Traub, "Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers", December 1974. AD A006868

Heller, D. E., "Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems", January 1975.

Hyafil, L. and H. T. Kung, "Parallel Algorithms for Solving Triangular Linear Systems with Small Parallelism", October 1974. AD A001376

Jones, A. K. and R. J. Lipton, "The Enforcement of Security Policies for Computation", May 1975.

Kacewicz, B., "An Integral-Interpolatory Iterative Method for the Solution of Non-Linear Scalar Equations", January 1975. AD A008811

Kacewicz, B., "The Use of Integrals in the Solution of Nonlinear Equations in N Dimensions", June 1975.

Karlton, P. L. and M. R. Barbacci, "IGRAPH: An Interactive Graphics Editor", September 1974.

For references by H. T. Kung, see R. Brent and L. Hyafil.

For references by V. R. Lesser, see L. D. Erman, R. D. Fennell and S. H. Fuller.

Lunde, A., "Evaluation of Instruction Set Processor Architecture by Program Tracing", Ph.D. Dissertation, July 1974. AD A004824

Newcomer, J. M., "Machine-Independent Generation of Optimal Local Code", Ph.D. Dissertation, May 1975.

Newell, A. and G. Robertson, "Some Issues in Programming Multi-Mini Processors" January 1975. AD A008858

Oakley, J., "A Comparison of Two Microprogrammable Processors: MLP-900 and PDP-11/40E", May 1975.

Ohlander, R., "Analysis of Natural Scenes", Ph.D. Dissertation, April 1975.

Perdue, C., "User's Introduction to UCI LISP", August 1974.

Schkolnick, M., "The Optimal Selection of Secondary Indices for Files", November 1974. AD A005692

Schkolnick, M., "On a Covering Problem for Partially Specified Switching Functions", December 1974. AD A005691

Shaw, M. and J. F. Traub, "Analysis of a Family of Algorithms for the Evaluation of a Polynomial and Some of Its Derivatives", February 1975. AD A009250

Shaw, M. (ed.), "IC Study Problems: Solution Collection", July 1974.

Shaw, M. and J. Buchanan, "Immigration Course in Computer Science", September 1974.

Siewiorek, D. P. and M. R. Barbacci, "Some Observations of Modular Design Technology and the Use of Microprogramming", July 1974. AD A004093

Siewiorek, D. P., "Reliability Modeling of Compensating Module Failures in Majority Voted Redundancy", October 1974. AD A004336

For other references by D. P. Siewiorek, see M. R. Barbacci.

Simon, H. A. and J. B. Kadane, "Optimal Problem-Solving Search: All-or-None Solutions", December 1974. AD A009141

Stickel, M. E., "The Programmable Strategy", July 1974. AD A004474

Stickel, M. E., "A Complete Unification Algorithm for Associative-Commutative Functions[1,2]", June 1975.

Swan, R. J. and S. H. Fuller, "K.mon: The C.mmp Hardware Monitor. A Programmer's Manual", June 1975.

Traub, J. F. (ed.), "Symposium on Analytic Computational Complexity Program and Abstracts", April 1975.

For other references by J. F. Traub, see A. Feldstein, D. E. Heller and M. Shaw.

Waterman, D. A., "Adaptive Production Systems", December 1974.

Waterman, D. A., "Serial Pattern Acquisition: A Production System Approach", February 1975.

Woźniakowski, H., "Numerical Stability for Solving Nonlinear Equations", February 1975. AD A006862

Woźniakowski, H., "Numerical Stability of the Chebyshev Method for the Solution of Large Linear Systems", March 1975. AD A006863

Woźniakowski, H., "Numerical Stability of Iterations for Solution of Nonlinear Equations and Large Linear Systems", June 1975.

Woźniakowski, H., "Maximal Order of Multipoint Iterations Using n Evaluations", July 1975.

Wulf, W. A., "ALPHARD: Toward a Language to Support Structured Programs", 1974.

Colloquia

September

John McCarthy, Stanford University, "Extensions of Predicate Calculus to Cover Common Sense Reasoning", September 12, 1974.

Harlan, D. Mills, International Business Machines, "How to Write Correct Programs and Know It", September 16, 1974.

James E. Thornton, Network Systems Corporation, "LSI for Super Computers—Problems and Issues", September 19, 1975.

James L. McKenney, Harvard University, "Five Unsolved Management Problems of Computer Systems", September 23, 1974.

William M. Kahan, University of California, "Do You Trust Your Calculator?", September 26, 1974.

October

Albert Meyer, Massachusetts Institute of Technology, "Some Computational Hopeless Problems", October 1, 1974.

Jurg Nievergelt, University of Illinois, "ACSES, An Automated Computer Science Education System", October 4, 1974.

Michael Rabin, Hebrew University, "Complexity of Computations: Its Uses and Misuses", October 8, 1974.

J. T. Godfrey, On-Line Systems, Inc., "The New Computer Service Industry", October 9, 1974.

Donald E. Knuth, Stanford University, "Fast Pattern Matching in Strings" and "Computer Programming as an Art", October 11, 1974.

Kenneth M. Colby, Stanford University, "Simulation of Paranoid Processes", October 15, 1974.

Thomas G. Price, Naval Postgraduate School, "Performance Models of Multiprogram Computer Systems", October 24, 1974.

November

J. W. de Bakker, Mathematics Center and Free University, Amsterdam, Lecture Series on Fixed Point Theory, November 11-21, 1974, 6 lectures presented.

Forest Baskett, Stanford University, "A Stochastic Model of Program Paging Behavior", November 20, 1974.

Richard Sites, Stanford University, "Proving That Computer Programs Terminate Cleanly", November 27, 1974.

December

James C. King, IBM T. J. Watson Research Center, "Program Testing Using Interactive Symbolic Execution", December 9, 1974.

Gerald J. Popek, University of California at Los Angeles, "Some Recent Activity in Protection", December 9, 1975.

Philip Meir Merlin, University of California at Irvine, "A Study of the Recoverability of Computing Systems", December 11, 1974.

Thomas Cheatham, Harvard University, "A Look at Programming Languages and Systems", December 11, 1974.

January

Peter Lauer, University of Newcastle-upon-Tyne, "An Extension of Methods for Guaranteeing Syntactic Correctness to Semantics of Concurrent Processes", January 16, 1975.

H. W. Lawson, Jr., Linköpings University, "Proposal for a System Machine" and "A Flexible Asynchronous Microprocessor", January 27-28, 1975.

L. Hyafil, IRIA, France, "Design of Optimal Merge on Direct Access Devices", January 29, 1975.

February

Andrew C. Yao, University of Illinois, "Two Problems in Computational Complexity", February 5, 1975.

Carver A. Mead, California Institute of Technology, "ESP, A Distributed Architecture LSI Machine", February 11, 1975.

Norihisa Suzuki, Stanford University, "Automatic Verification of Programs", February 12, 1975.

Klaus Berkling, Gesellschaft fuer Mathematik und Dataverarbeitung, "Reduction Languages for Reduction Machines", February 13, 1975.

David Dobkin, Yale University, "On Some Fast Algorithms for Matrix Multiplication", February 14, 1975.

Michael Fischer, Massachusetts Institute of Technology, "Design of Small Logical Networks from Fast Turing Machines", February 26, 1975.

March

Michael Harrison, University of California at Berkeley, "Protection Systems", March 3, 1975.

Patricia Griffiths, Harvard University, "SYNVER: A System for the Automatic Synthesis and Verification of Synchronous Processes", March 5, 1975.

Chul E. Kim, University of Minnesota, "Approximation Algorithms for Some NP-Complete Problems", March 19, 1975.

Leonidas Guibas, Stanford University, "The Analysis of Double Hashing", March 11, 1975.

Douglas Jensen, "Distributed Processing in a Real-Time Environment", March 13, 1975.

Irene Greif, Massachusetts Institute of Technology, "Semantics of Communicating Parallel Processes", March 17, 1975.

David Milgram, University of Maryland, "Picture Processing with Histograms", March 18, 1975.

Michael Shamos, Yale University, "Fast Geometric Algorithms", March 19, 1975.

Raphael Rom, University of Utah, "Image Coding Based on Human Vision", March 31, 1975.

April

John Guttag, University of Toronto, "Specification of Abstract Data Types", April 2, 1975.

A. V. Aho, Bell Laboratories, "Compiler Compilers and Code Generation for Expressions", April 7, 1975.

William Lynch, Case Western Reserve University, "Performance Measurements at CWRU: The PDP-10/TOPS-10 System", April 9, 1975.

Gerald Belpaire, University of Wisconsin, "Toward a Synthesis of Synchronization Problems", April 9, 1975.

Brian Randall, University of Newcastle-upon-Tyne, "Computing System Reliability", April 11, 1975.

Ian Munro, University of Waterloo, "Searching and Sorting in Multisets", April 21, 1975.

Marvin Solomon, Cornell University, "The Semantics of Data Types", April 22, 1975.

Theodore H. Kehl, University of Washington, "A Logic Machine Multiprocessor", April 24, 1975.

John Darlington, University of Edinburgh, "Transforming High Level Programs", April 25, 1975.

W. M. Turski, University of Warsaw, "Informatics (Computing Science?)—As A Natural Science", April 29, 1975.

May

Yuri Breitbart, Technion-Israel Institution of Technology, "Analysis of Algorithms for the Evaluation of Monotonic Boolean Function", May 5, 1975.

Erik Sandehall, Uppsala University, Visiting Professor at MIT, "Structured Programming a la Artificial Intelligence", May 7, 1975.

Axel van Lamsweerde, MBLE Research Laboratory, "Derivation of Correct and Efficient Concurrent Programs", May 13, 1975.

Gifts, Grants, And Contracts

Advanced Research Projects Agency, Prof. A. Newell and Prof. J. F. Traub, Research in Information Processing, July 1, 1974—June 30, 1975.

Alcoa Foundation, Department, 2 partial scholarships, 1974-75.

Defense Communication Agency, Prof. S. H. Fuller, The Application of Multiple Processor Computer Systems to Digital Communications Networks, June 30, 1975.

Ford Motor Company, Prof. W. A. Wulf, unrestricted, 1974-75.

International Business Machines, Department, 1 full fellowship, 1974-75.

International Business Machines, Department, unrestricted, 1974-75.

Mellon Computer Research Funds, Department, Computer Research, unrestricted.

National Science Foundation, Prof. J. F. Traub, Research in Parallel Algorithms and Computational Complexity, January 15, 1972—December 31, 1975.

National Science Foundation, Prof A. N. Habermann, Type Definitions and Path Expressions as Design Tools, April 15, 1975—June 30, 1975.

National Science Foundation, Prof. A. N. Habermann, System Family Concept, July 1, 1972—December 31, 1974.

National Science Foundation, Prof. A. Jones, A Formal Theory of the Controlled Dissemination of Information, January 1, 1975—June 30, 1975.

National Science Foundation, Prof. M. Shaw, Software to Support Well-Structured Programming, November 15, 1974—June 30, 1975.

National Science Foundation, Prof. D. P. Siewiorek, Register Transfer Level Computer Systems Design, July 1, 1972—June 30, 1975.

Office of Naval Research, Prof. J. F. Traub, Analysis of Algorithms, July 1, 1971—August 31, 1975.

Rome Air Development Center, Prof. M. Shaw, Tools for Good Structure, March 1, 1975—June 30, 1975.

Shell Companies Foundation, Department, Computer Science Research, 1974-75.

Xerox Corporation, Prof. A. Newell, Research in Cognitive Processes, 1974-1975.

Xerox Corporation, Prof. D. R. Reddy, 1 full fellowship, 1974-75.

Ph.D. Dissertations

The following persons have been awarded Ph.D.'s in Computer Science and related areas since the establishment of the Computer Science Department in 1965. The department or program from which each received his Ph.D. is followed by his most recent position.

The accession numbers follow in parentheses after those dissertations which are registered as reports with the Defense Documentation Center.

Aygun, Birol (Computer Science), Staff Programmer, IBM Corporation Advanced Systems Development Division, Yorktown Heights, New York, "Dynamic Analysis of Execution Possibilities, Techniques and Problems", 1974, Professor J. W. McCredie.

Baker, James K. (Computer Science and Speech), IBM Thomas J. Watson Research Center, Yorktown Heights, New York, "Stochastic Modeling as a Means of Automatic Speech Research", 1975, Professor R. Reddy.

Baker, Janet M. (Biocommunication and Computer Science), IBM Thomas J. Watson Research Center, Yorktown Heights, New York, "A New Time-Domain Analysis of Human Speech and Other Complex Wave Forms", 1975, Professor R. Reddy.

Balzer, Robert M. (Systems and Communication Sciences), Staff Member, University of Southern California, Information Sciences Institute, Marina Del Rey, California, "Studies Concerning Minimal Time Solutions to the Firing Squad Synchronization Problem", 1966, Professor A. Newell. (AD 635056)

Barbacci, Mario R. (Computer Science), Research Associate, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Automated Exploration of the Design Space for Register Transfer Systems", 1974, Professor C. G. Bell.

Berglass, Gilbert R. (Systems and Communication Sciences), Assistant Manager, Computing Center, State University of New York at Buffalo, Buffalo, New York, "A Generalization of Macro Processing", 1970, Professor A. J. Perlis.

Berliner, Hans J. (Computer Science), Research Associate, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Chess as Problem Solving: The Development of a Tactics Analyzer", 1975, Professor A. Newell. (AD 784881)

Caviness, B. F. (Mathematics), Assistant Professor of Computer Science, University of Wisconsin, Madison, Wisconsin, "On Canonical Forms and Simplification", 1967, Professor A. J. Perlis. (AD 671938)

Charon, Robert N. (Computer Science), Assistant Professor of Computer Science, Department of Computer Science, University of Texas, Austin, Texas, "On a Measure of Program Structure", 1974, Professor D. L. Parnas.

Chen, Robert C. (Computer Science), Assistant Professor, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, Pennsylvania, "Bus Communications System", 1974, Professor C. G. Bell.

Coles, L. Stephen (Systems and Communication Sciences), Senior Research Mathematician, Stanford Research Institute, Menlo Park, California, "Syntax Directed Interpretation of Natural Language", 1967, Professor H. A. Simon. (AD 655923)

Darringer, John A. (Systems and Communications Sciences), IBM T. J. Watson Research Center, Yorktown Heights, New York, "The Description, Simulation, and Automatic Implementation of Digital Computer Processors", 1969, Professor D. L. Parnas. (AD 700144)

Earley, Jay (Computer Science), Research Associate, Department of Computer Science, University of California, Berkeley, California, "An Efficient Context-Free Parsing Algorithm", 1968, Professor R. W. Floyd.

Ernst, George (Systems and Communication Sciences), Associate Professor, Department of Computer Science, Computer Engineering Division, Case Western Reserve University, Cleveland, Ohio, "Generality and GPS", 1966, Professor A. Newell. (AD 809354)

Evans, Arthur, Jr. (Mathematics), Lincoln Laboratory, Lexington, Massachusetts, "Syntax Analysis by a Production Language", 1965, Professor A. J. Perlis. (AD 625465)

Farley, Arthur, M. (Computer Science [Systems and Communication Sciences]), "VIPS: A Visual Imagery and Perception System; the Results of a Protocol Analysis", 1974, Professor A. Newell.

Feldman, Jerome A. (Mathematics), Professor and Chairman, Department of Computer Science, University of Rochester, Rochester, New York, "A Formal Semantics for Computer Oriented Languages", 1964, Professor A. J. Perlis. (AD 462935)

Fennell, Richard D. (Computer Science), Systems Programmer, Judicial Court System, Washington, D.C., "Multiprocess Software Architecture for AI Problem Solving", 1975, Professor R. Reddy.

Fikes, Richard E. (Computer Science), Senior Research Mathematician, Stanford Rsearch Institute, Menlo Park, California, "A Heuristic Program for Solving Problems Stated as Nondeterministic Procedures", 1969, Professor A. Newell. (AD 688604)

Fisher, David (Computer Science), Member of Research Staff, Science and Technology Division, Institute for Defense Analyses, Arlington, Virginia, "Control Structures for Programming Languages", 1970, Professor A. J. Perlis. (AD 708511)

Freeman, Peter A. (Computer Science), Assistant Professor of Systems and Information Science, University of California, Irvine, California, "Sourcebook for OSD—An Operating System Designer", 1970, Professor A. Newell.

Gerhart, Susan L. (Computer Science), Duke University, Durham, North Carolina, "Verification of APL Programs", 1973, Professor D. Loveland. (AD 754856)

Geschke, Charles M. (Computer Science), Xerox Research Center, Palo Alto, California, "Global Program Optimization", 1973, Professor W. Wulf.

Gibbons, Gregory D. (Computer Science), System Control Incorporated, Palo Alto, California, "Beyond REF-ARF: Toward an Intelligent Processor for a Nondeterministic Language", 1973, Professor A. Newell. (AD 755811)

Grason, John (Electrical Engineering [Systems and Communication Sciences]), Assistant Professor of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Methods for the Computer-Implemented Solution of a Class of 'Floor Plan' Design Problems", 1970, Professor H. A. Simon. (AD 717756)

Haney, Frederick M. (Computer Science), Manager, Technical Integration and Planning, Xerox Corporation, El Segundo, California, "Using a Computer to Design Computer Instruction Sets", 1968, Professor C. G. Bell. (AD 671939)

Hansen, Gilbert J. (Computer Science), Assistant Professor, Department of Computer and Information Science, University of Florida, Gainesville, Florida, "Adaptive Systems for the Dynamic Run-Time Optimization of Programs", 1974, Professor W. A. Wulf.

Huen, Wing Hing (Computer Science), Visiting Assistant Professor, Illinois Institute of Technology, Chicago, Illinois, "A Unifying Notation and Analysis of Modular, Register Transfer (RT) Control", 1974, Professor C. G. Bell.

Iturriaga, Renato (Computer Science), Director of the Computation Center and of the Center for Research on Applied Mathematics, University of Mexico, Mexico City, "Contributions to Mechanical Mathematics", 1967, Professor A. J. Perlis. (AD 660127)

Jones, Anita (Computer Science), Assistant Professor, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Protection in Programmed Systems", 1974, Professor A. N. Habermann. (AD 765535)

King, James C. (Computer Science), Research Staff, IBM T. J. Watson Research Center, Yorktown Heights, New York, "A Program Verifier", 1970, Professor R. W. Floyd. (AD 699248)

Knudsen, Michael J. (Computer Science), Bell Laboratories, Naperville, Illinois, "PMSL, An Interactive Language for System-Level Description and Analysis of Computer Structures", 1973, Professor C. G. Bell.

Ladron DeCegama, Angel (Electrical Engineering [Systems and Communication Sciences]), Senior Research Engineer, National Cash Register Company, Hawthorne, California, "Performance Optimization of Multiprogramming Systems", 1970, Professor A. J. Perlis.

Lauer, Hugh C. (Computer Science), Lecturer, Computing Laboratory, University of Newcastle, Newcastle Upon Tyne, England, "Correctness in Operating Systems", 1973, Professor W. A. Wulf. (AD 753122)

Lindstrom, Gary (Computer Science), Assistant Professor of Computer Science, University of Pittsburgh, Pittsburgh, Pennsylvania, "Variability in Language Processors", 1970, Professor A. J. Perlis. (AD 714695)

Lipton, Richard (Computer Science), J. Willard Gibbs, Instructor, Department of Computer Science, Yale University, New Haven, Connecticut, "On Synchronization Primitive Systems", 1974, Professor D. L. Parnas. (AD 764782)

London, Ralph L. (Mathematics), Information Sciences Institute, University of Southern California, Marina Del Rey, California, "A Computer Program for Discovering and Proving Sequential Recognition Rules for Well-Formed Formulas Defined by a Backus Normal Form Grammar", 1964, Professor A. Newell. (AD 840036)

Lunde, Amund, (Computer Science), University of Oslo, Norway, "Evaluation of Instruction Set Processor Architecture by Program Tracing", 1975, Professor W. Wulf. (AD A004824)

Mann, William C. (Computer Science), Project Leader, University of Southern California, Information Sciences Institute, Marina Del Ray, California, "Memory Processes for Information Assimilation", 1974, Professor A. Newell.

Manna, Zohar (Computer Science), Associate Professor, Applied Mathematics Department, Weizmann Institute of Science, Rehovot, Israel, "Termination of Algorithms", 1968, Professor R. W. Floyd. (AD 670558)

McCredie, John W. (Systems and Communication Sciences from Graduate School of Industrial Administration), Director of Computation Services, Lecturer in Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Analytic Models of Time-Shared Computing Systems: New Results, Validations, and Uses", 1972, Professor C. Kriebel.

McCreight, Edward M. (Computer Science), Research Scientist, Xerox Research Center, Palo Alto, California, "Classes of Computable Functions Defined by Bounds on Computation", 1970, Professor A. R. Meyer. (AD 693327)

Mitchell, James G. (Computer Science), Xerox Research Center, Palo Alto, California, "The Design and Construction of Flexible and Efficient Interactive Programming Systems", 1970, Professor A. J. Perlis. (AD 712721)

Moore, James A. (Computer Science), University of Southern California, Information Sciences Institute, Marina Del Ray, California, "The Design and Evaluation of a Knowledge Net for MERLIN", 1971, Professor A. Newell.

Moran, Thomas P. (Computer Science), Research Scientist, Xerox Research Center, Palo Alto, California, "The Symbolic Imagery Hypothesis: An Empirical Investigation via a Product System Simulation of Human Behavior in a Visualization Task", 1974, Professor A. Newell.

Mullin, James K. (Systems and Communication Sciences), Associate Professor, Computer Science Department, University of Western Ontario, London, Ontario, Canada, "A Computer Optimized Question Asker for Aiding Bacteriological Species Identification COQAB", 1967, Professor B. Green.

Newcomer, Joseph M. (Computer Science), Research Associate, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Machine-Independent Generation of Optimal Local Code", 1975, Professor W. Wulf.

Ohlander, Ronald B. (Computer Science), Chief Engineer, U.S. Navy, "Analysis of Natural Scenes", 1975, Professor R. Reddy.

Parnas, David L. (Systems and Communication Sciences), Professor, Technical University of Darmstadt, West Germany, "System Function Description ALGOL—A Language for the Description of the Functions of Finite State Systems, the Simulation of Finite Systems, and the Automatic Production of the State Tables of Such Systems" 1965, no advisor. (AD 467633)

Pfefferkorn, Charles (Computer Science), ILLIAC-IV Project Manager, Evans and Sutherland Computer Corp., Sunnyvale, California, "Computer Design of Equipment Layouts Using the Design Problem Solver (DPS)", 1971, Professor H. A. Simon.

Price, William R. (Computer Science), Computer Systems Design Engineer, Hq. ESD/MCI, L. G. Hanscom Field, Massachusetts, "Virtual Memory Mechanism for Implement Protection in a Family of Operating Systems", 1974, Professor D. L. Parnas. (AD 766292)

Quatse, Jesse T. (Electrical Engineering and Systems and Communication Sciences), Telemechanique, Rue De Provence, 38 Echirolles, France, "A Highly-Modular Organization of General Purpose Computers", 1969, Professor C. G. Bell.

Quillian, M. Ross (Psychology), Associate Professor, Social Sciences Department, University of California, Irvine, California, "Semantic Memory", 1967, Professor H. A. Simon.

Reeker, Larry H. (Systems and Communication Sciences), Assistant Professor, Oregon University, Eugene, Oregon, "A Problem-Solving Theory of Syntactic Acquisition", 1974, Professor A. Newell.

Richardson, Leroy (Systems and Communication Sciences), Staff Scientist, Information Sciences Institute, University of Southern California, Marina Del Rey, California, "Specification Techniques for Interactive Computer Systems", 1972, Professor D. L. Parnas.

Shaw, Mary M. (Computer Science), Assistant Professor of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Language Structures for Contractible Compilers", 1972, Professor A. J. Perlis. (AD 744117)

Shoup, Richard (Computer Science), Xerox Research Center, Palo Alto, California, "Programmable Cellular Logic Arrays", 1970, Professor C. G. Bell. (AD 706891)

Siklossy, Laurent (Computer Science), Computer Sciences Department, University of Texas, Austin, Texas, "Natural Language Learning by Computer", 1968, Professor H. A. Simon. (AD 671937)

Snyder, Lawrence (Computer Science) Assistant Professor of Computer Science, Yale University, New Haven, Connecticut, "An Analysis of Parameter Evaluation for Recursive Procedures", 1973, Professor A. N. Habermann.

Standish, Thomas A. (Computer Science), Department of Information and Computer Science, University of California, Irvine, California, "A Data Definition Facility for Programming Languages", 1967, Professor A. J. Perlis. (AD 658042)

Strauss, Jon C. (Systems and Communication Sciences), Professor and Director of Computing Activities, Office of Computing Activities, University of Pennsylvania, Philadelphia, Pennsylvania, "Identification of Continuous Dynamic Systems by Parameter Optimization", 1965, Professor A. Lavi. (AD 660887)

Strecker, William D. (Electrical Engineering), Research and Development Group, Digital Equipment Corporation, Maynard, Massachusetts, "An Analysis of the Instruction Execution Rate in Certain Computer Structures", 1970, Professor C. G. Bell. (AD 711408)

Wagner, Robert A. (Computer Science), Associate Professor, Department of Systems and Information Science, Vanderbilt University, Nashville, Tennessee, "Some Techniques for Algorithm Optimization with Application to Matrix Arithmetic Expressions", 1969, Professor A. J. Perlis. (AD 678629)

Waldinger, Richard J. (Computer Science), Research Mathematician, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, "Constructing Programs Automatically Using Theorem Proving", 1969, Professor H. A. Simon. (AD 697061)

Wile, David D. (Computer Science), Staff Member, University of Southern California, Information Sciences Institute, Marina Del Rey, California, "A Generative, Nested Sequential Basis for General Purpose Programming Languages", 1974, Professor W. A. Wulf. (AD 773839)

Williams, Donald S. (Systems and Communication Sciences), Member Technical Staff, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, "Computer Program Organization Induced by Problem Example", 1969, Professor H. A. Simon. (AD 688242)

Winikoff, Arnold W. (Systems and Communication Sciences), President, Q.E.D., Inc., Minneapolis, Minnesota, "Eye Movement as an Aid to Protocol Analysis of Problem Solving Behavior", 1967, Professor A. Newell.